EXCEL SCHNELLER,
RASCHER,
SICHERER
MIT
VISUAL BASIC FÜR ANWENDUNGEN

© 1996 BY JOSEF BROUKAL , VIENNA

© by Josef Broukal, Vienna, Austria

Inhalt

Inhalt i

Wozu Sie Visual Basic für Anwendungen in EXCEL verwenden können 1
VBA–Programme helfen Ihnen… 1
Fragen, auf die dieses Skriptum Antwort gibt: 1

Das erste Programm 3
1. Schritt: Starten Sie den Makro–Rekorder 3
2. Geben Sie EXCEL einige Informationen… 4
3. Zeigen Sie EXCEL, was es tun soll 5
4. Testen Sie den neuen Makro 5
5. Nachbearbeiten 7
Die Aufzeichnung nachbessern 7
Damit Sie nicht zu fröhlich werden 8
So starten Sie VBA–Programme 9

VBA-Programme formal gesehen 10

Nützen Sie die Online-Hilfe! 12
Online-Hilfe ergänzen 13

Die Symbolleiste „Visual Basic“ 14
Elemente der Symbolleiste „Visual Basic“ 14
Das VBA–Testfenster 15
Programme schrittweise abarbeiten 16
Haltepunkt setzen 17
Direktbereich benützen 17
Weitere Möglichkeiten zur Arbeit mit dem Testfenster 22

Sprachbestandteile von VBA 24
Prozeduren 24
Funktionen 28
Gültigkeitsbereich von Prozeduren und Funktionen 30
Anweisungen 31
Variablen und Datentypen 31
Deklaration und Gültigkeitsbereich von Variablen 33
Statische Variable 34
Datenfelder 34
Die Anweisung „Option Base“ 34
Beispiele 35
Gültigkeitsbereich von Datenfeldern 36
Die „Array“–Funktion 36
Anmerkungen 36
Datenfeld-Variant mit Inhalt eines Zellbereiches füllen 37
Konstanten 38
Deklaration von Konstanten 38
Operatoren 39
Arithmetische Operatoren 39
Logische Operatoren 39
Vergleichsoperatoren 39
Verkettungsoperatoren 40

© by Josef Broukal, Vienna, Austria

Objekte **40**
Gültigkeitsbereich von Objekten 40
Schlüsselwörter nach Aufgabenbereichen **40**

Objekte 1: Übersicht **42**
Übersicht 42
Eigenschaften von Objekten 44
Durchführen von Aktionen mit Objektmethoden 45
Objekt „Application“ („Anwendung“) 48
Container-Objekt „Workbooks“ („Arbeitsmappenliste“) 50
Objekt „Workbook“ („Arbeitsmappe“) 50
Container-Objekt „Windows“ („Fensterliste“) 50
Objekt „Window“ (Fenster“) 52

Objekte 2: Zellen und Zellbereiche **54**
Bereiche aus mehreren Zellen 54
Sonderfall: Bereich umfaßt nur eine Zelle 56
Wie kommen wir zur Zelle B5 des Arbeitsblattes „Tabelle1“? 57
Eigenschaften von Zelle B5 setzen und erfragen 58
Eigenschaften mehrerer Zellen setzen und erfragen 58

Zum Start zwei kurze Beispiele… **60**
Automatisch Zoomen 60
Spalten so breit wie ihr Inhalt machen 61

Steuern des Programmablaufs **63**
Bedingtes Ausführen von Codeabschnitten **63**
If…Then („Wenn…Dann“) 63
If…Then…ElseIf…Else…End If(„Wenn…Dann…Sonst“) 64
Select Case 65
Mehrfaches Ausführen desselben Codes mit Programmschleifen **66**
Do While…Loop 66
For…Next 68
For Each…Next 69
Schleifen vorzeitig verlassen 71
Automatisches Ablaufen von Programmen **72**
Beim Laden einer Arbeitsmappe 72
Beim Aktivieren und Verlassen eines Tabellenblattes 73

Einzelfragen **75**
Feststellen, wo der Cursor steht **75**
Den Cursor versetzen **75**
Feststellen, ob umliegende Zellen beschrieben worden sind **76**
Feststellen, welche Zellen markiert worden sind **79**
Ein Bereich 79
Mehrere Bereiche 79
Auf Arbeitsblättern navigieren **80**
Ausschneiden, Kopieren und Einfügen **82**
Den Inhalt von Zellen ausschneiden—„Cut“ 83
Den Inhalt von Zellen kopieren—„Copy“ 83
Den Inhalt der Zwischenablage einfügen—„Paste“ 84
Für das Einzufügende Platz machen—„Insert“ 84
Ein Bereichs–Objekt erzeugen **84**
Grösse eines Bereichs–Objektes ändern **87**
Verschieben eines Bereichs–Objektes **88**

© by Josef Broukal, Vienna, Austria

Eine Meldung auf dem Bildschirm anzeigen 89
„MsgBox“—Programm–unterbrechende Meldungen 89
Nicht–unterbrechende Meldung 91
Nicht–unterbrechende Meldung in EXCEL–Tabelle 92
Eine Eingabe durch den Benützer erhalten 92
Dateien kontrolliert öffnen lassen 96
Dateien kontrolliert speichern 98
Eine Anwendung vor unbefugtem Zugriff schützen 99
Zellen 99
Blätter 100
Arbeitsmappen 101
Teile einer Anwendung ausblenden 102
Blätter 103
Mappen 103
Eine Formel in eine Tabelle eintragen 104
Namen verwenden 104

Daten filtern mit dem „Autofilter“ 108
Die Methode „Autofilter“ 108
Die Eigenschaft „AutoFilterMode“ 109
Die Eigenschaft „FilterMode“ 110
Daten filtern 110

Menüs mit VBA verändern 114
Befehle 114
Einen eingebauten Befehl löschen 114
Einen eingebauten Befehl wiederherstellen 115
Einen eigenen Befehl einfügen 115
Einen eigenen Befehl löschen 116
Zwischenstrich einfügen 116
Zwischenstrich löschen 117
Menüs 117
Menü in Menüleiste einfügen 117
Menü löschen 118
Untermenü einfügen 118
Kontextmenüs 118
Kontextmenü erweitern 118
Kontextmenü löschen 119
Ausprogrammiertes Beispiel 119

Selbstgestaltete Dialogfelder 122
Die Symbolleiste „Dialog“ 122
Elemente eines Dialogfeldes 123
Der lange Weg zum fertigen Dialogfeld 123
Zeichnen des Dialogfeldes 123
Reihenfolge der Elemente festlegen 124
Code für das Füllen der Elemente vor dem Anzeigen des Dialogfeldes schreiben 125
Code für die laufende Kontrolle des Dialoges schreiben 125
Dialogfeld-Elemente im Überblick 129

Host-Daten einlesen 138
Sonderfälle 138
Datum als Zeichenkette ohne Trennzeichen 138
Kennziffern mit führenden Nullen 138

© by Josef Broukal, Vienna, Austria

EXCEL Makros und VBA — einmal hin und zurück 140
VBA-Äquivalente der EXCEL 4.0 Makrosprache 141

Bücher und andere Hilfen 142
Bücher 142
Hilfe Online 142
Hilfe durch J. Broukal 143

© by Josef Broukal, Vienna, Austria

Wozu Sie Visual Basic für Anwendungen in EXCEL verwenden können

EXCEL wird zusammen mit Visual Basic für Anwendungen (VBA) ausgeliefert, einer Programmiersprache, die das Arbeiten mit Tabellen, Daten und Diagrammen vereinfacht und beschleunigt. VBA ist allerdings eine „ausgewachsene“, sehr umfangereiche Angelegenheit. Sie zu erlernen, ist mühsam; sie zu beherrschen, erfordert ständige Übung. Der Lohn der Mühe sind mächtige Programme, die viel Zeit ersparen und viel Sicherheit beim Umgang mit Tabellen bringen.

EXCEL wird mit einer ausführlichen Online-Hilfe ausgeliefert, die in Form einer alphabetischen Referenz (fast) alles enthält, was man über Visual Basic für Anwendungen wissen muß. Diese neue Sprache enthält tausende Elemente. Sie vollständig zu behandeln, würde den Rahmen dieses Skriptums sprengen. Was es kann ist, jene Arbeitsanweisungen vorzustellen, die für die Steuerung von VBA–Programmen und für das Schreiben von neuen EXCEL-Funktionen gebraucht werden. Es soll Sie in die Lage versetzen, das Grundgerüst eines VBA–Programmes zu erstellen, in das Sie dann jene Anweisungen einfügen, die die konkrete Verarbeitung Ihrer Daten erledigen.

VBA–Programme helfen Ihnen…

- immer wiederkehrende Arbeiten automatisch zu erledigen
- zu den von EXCEL zur Verfügung gestellten Funktionen neue hinzuzufügen
- wenig geübten EXCEL-Benützern eine Arbeitsumgebung zu geben, in der nur noch jene Menüs und Befehle vorhanden sind, die für die konkrete Arbeit gebraucht werden
- EXCEL-Anwendungen herzustellen, die den Benützer genauso durch eine Arbeitsfolge führen wie ein mit einer Programmiersprache erstelles Programm
- von EXCEL aus andere Microsoft–Anwendungen zu starten und auszuführen. Microsoft hat versprochen, nach und nach alle „Office“–Produkte mit VBA auszurüsten. Projekt 4.0 hat bereits VBA, PowerPoint für Windows 95 ebenfalls, und beide können daher von EXCEL aus *mit der selben Programmiersprache* gesteuert werden.

In EXCEL für Windows 95 ist VBA in englischer Sprache implementiert. Leider gibt es kein englisches Verzeichnis der Tabellenfunktionen. Sie sind in der Online-Hilfe nur deutsch enthalten, obwohl VBA sie im Code auf Englisch erwartet. Hier hilft nur: Tabellenblatt hernehmen, Makrorekorder starten, Funktion in eine Zelle eintragen, aufgezeichneten Code in Ihr Programm übertragen.

Fragen, auf die dieses Skriptum Antwort gibt:

- Wie baue ich eine VBA–Programm auf?
- Wie dokumentiere ich ein VBA–Programm?
- Wie programmiere ich eine Schleife?
- Wie verzweige ich in ein anderes VBA–Programm in Abhängigkeit von einer Bedingung?
- Wie weiß ich, auf welche Datei und Zelle sich mein VBA–Programm bezieht?
- Wie schreibe ich eine neue EXCEL-Funktion?
- Wie finde und beseitige ich Programm–Fehler?
- Wie ersetze ich das EXCEL-Menü durch ein eigenes?
- Wie kann ich ein VBA–Programm automatisch beim Aufruf von EXCEL starten lassen?
- Wie kann ich ein VBA–Programm immer dann starten, wenn eine Tabelle geöffnet wird?
- Wie kann ich VBA–Programm immer dann starten, wenn eine Tabelle geschlossen wird?
- Wie schütze ich Dateien und Zellen vor dem Zugriff durch den Benutzer und anderer Leute neugierige Augen, kann aber dennoch per Makro in sie Daten eingeben und Verarbeitungen vornehmen?

© by Josef Broukal, Vienna, Austria

Wenn Sie mit diesem Skriptum arbeiten wollen, sollten Sie mit den Grundbegriffen von EXCEL und WINDOWS vertraut sein.

© by Josef Broukal, Vienna, Austria

Das erste Programm

In diesem Kapitel lernen Sie:

- Bedienen des Makro–Rekorders
- Entfernen unnötig aufgezeichnetes Codes
- Nachaufzeichnen einzelner Befehle
- Speichern des Makros

Programme, die wie ein aufgezogenes Uhrwerk ablaufen—ohne Wenn und Aber—können Sie EXCEL vorzeigen. Der „Makro–Rekorder“ schreibt mit. Ein solches Programm wollen wir als erstes erstellen. Es soll beim Ausdrucken von Tabellen

- die Zeilen- und Spaltenköpfe ausblenden
- die Gitternetzlinien nicht drucken

Die entsprechenden Schalter finden Sie im Menü „Datei“ im Befehl „Seite einrichten…“. Wir wollen sie betätigen, während der Makro-Rekorder mitschreibt.

1. Schritt: Starten Sie den Makro–Rekorder

Öffnen Sie das Menü „Extras“. Wählen Sie den Befehl „Makro aufzeichnen…“ und im Untermenü den Befehl „Aufzeichnen…“

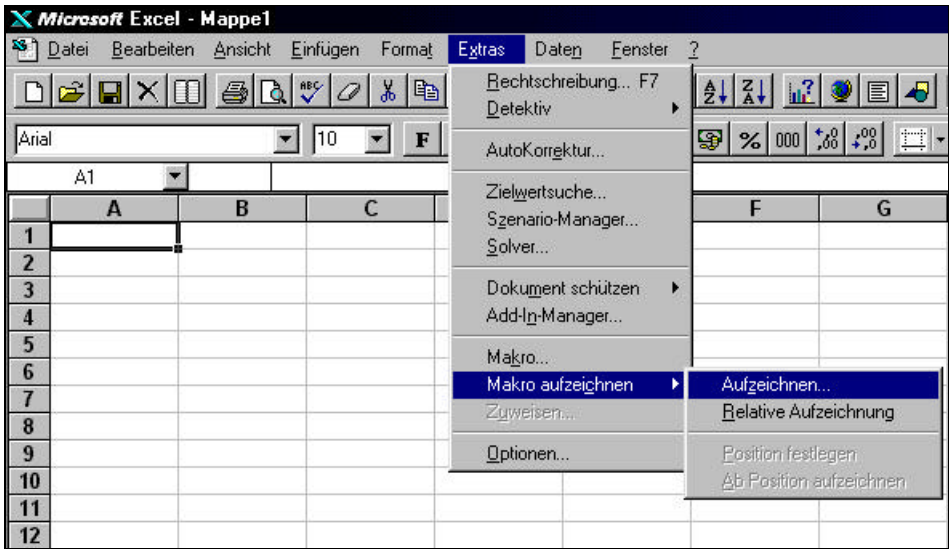


Abbildung 1: So finden Sie den Befehl „Aufzeichnen…“ im EXCEL-Menü „Extras“

© by Josef Broukal, Vienna, Austria

2. Geben Sie EXCEL einige Informationen…

EXCEL fragt Sie zunächst nach einigen Informationen über den aufzuzeichnenden Makro:

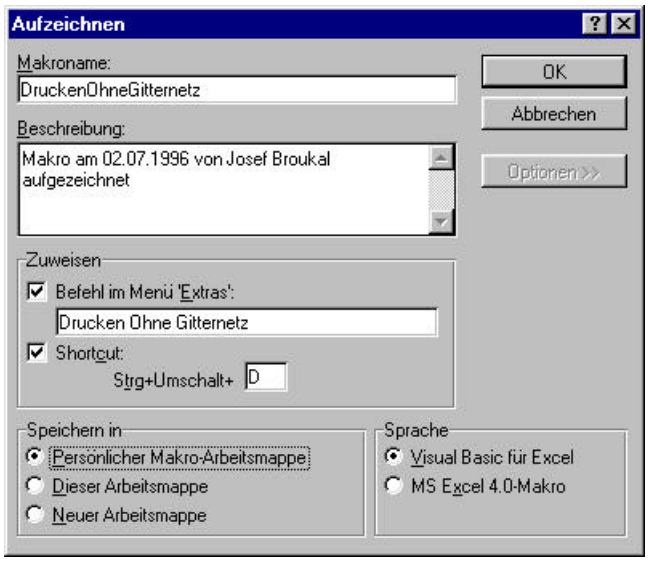


Abbildung 2: Im Dialogfeld „Aufzeichnen“ legen Sie nähere Einstellungen zu Ihrem Makro fest

Makroname EXCEL gibt die Bezeichnung „Makro“ gefolgt von einer automatisch angefügten Zahl (1…n) vor. Ersetzen Sie diesen Namen durch „DruckenOhneGitternetz“.

Beschreibung EXCEL gibt den in der Abbildung gezeigten Text vor; es verwendet dafür das Computer–Systemdatum und den bei der Installation von EXCEL angegebenen Benutzernamen.

Befehl im Menü ‘Extras’ Schreiben Sie in dieses Bearbeitungsfeld den Text „DruckenOhneGitternetz“.

Shortcut Setzen Sie in dieses Feld den Buchstaben „D“. Wenn Sie nach dem Aufzeichnen des Makros später STRG+D drücken, wird EXCEL diesen Makro starten. EXCEL unterscheidet übrigens beim Aufrufen von Makros zwischen Groß- und Kleinbuchstaben. Genau genommen müssen Sie daher STRG+UMSCHALT+D drücken!

Speichern in legt fest, wo Ihr Makro gespeichert werden soll:

- **Persönlicher Makro–Arbeitsmappe:** Diese Arbeitsmappe erwartet EXCEL unter dem Namen „PERSONL.XLS“ im Verzeichnis „XLSTART“. Findet EXCEL diese Datei, dann wird sie beim Starten von EXCEL, wie jede in diesem Verzeichnis stehende Datei mit der Endung „XL?“, geladen; alle in ihr enthaltenen Makros stehen daher automatisch zur Verfügung. Findet EXCEL die Datei nicht, dann legt es sie an. Achtung: Die Datei wird sofort verborgen, Sie können Sie über das Menü „Fenster“ und den Befehl „Einblenden“ sichtbar machen.
- **Dieser Arbeitsmappe:** Der Makro wird in der gerade aktiven Arbeitsmappe gespeichert. Er ist später nur dann verfügbar, wenn die Mappe geladen worden ist
- **Neuer Arbeitsmappe:** EXCEL legt den Makro in eine neue Arbeitsmappe. Er ist später nur dann verfügbar, wenn die Mappe geladen worden ist. Wählen Sie diese Option.

Sprache Irgendwann, sagt Microsoft, wird die alte Makrosprache nicht mehr verwendet werden können. Bis dahin läßt man Ihnen beim Aufzeichnen eines Makros noch die Wahl zwischen

- **Visual Basic für EXCEL:** Die neue Makrosprache, die Sie lernen wollen—wählen Sie daher diese Option;

© by Josef Broukal, Vienna, Austria

- **MS EXCEL 4.0 Makro:** Die alte Makrosprache, umständlich, unstrukturiert und schwer zu durchschauen.

Schließen Sie diesen Arbeitsschritt durch Drücken der ENTER–Taste oder Mausklick auf den OK–Schaltknopf ab. In der Statuszeile, am linken unteren Rand des EXCEL–Fensters, zeigt Ihnen EXCEL mit dem Wort „Aufzeichnung“, daß es auf Ihre Befehle wartet.

3. Zeigen Sie EXCEL, was es tun soll

Ab jetzt schreibt EXCEL alles mit, was Sie mit Maus oder Tastatur vorzeigen. Führen Sie die folgenden Arbeitsschritte aus:

- Öffnen Sie das Menü „Datei“
- Wählen Sie den Befehl „Seite einrichten…“
- Wählen Sie im Dialogfeld „Seite einrichten“ das Register „Tabelle“
- Deaktivieren Sie das Kontrollkästchen „Gitternetzlinien“
- Aktivieren Sie das Kontrollkästchen „Schwarzweißdruck“
- Schließen Sie durch Drücken der ENTER–Taste oder Mausklick auf den OK–Schaltknopf das Bearbeiten des Dialogfeldes ab
- Stoppen Sie über das Menü „Makro“ und den Befehl „Makro aufzeichnen“ mit „Aufzeichnung beenden“ den Makro–Rekorder.

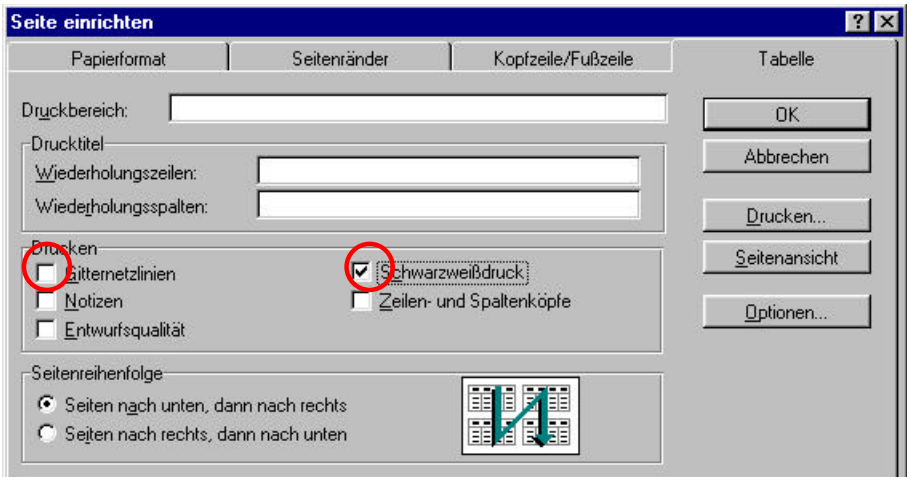


Abbildung 3: Diese beiden Kontrollkästchen müssen Sie ein- bzw. ausschalten.

Fertig. Jetzt wollen wir den Makro „DruckenOhneGitternetz“ testen.

4. Testen Sie den neuen Makro

Alles, was wir dazu brauchen, sind einige beschriebene Zellen auf einem Tabellenblatt. Öffnen Sie dann das Menü „Extras“ und wählen Sie den Befehl „DruckenOhneGitternetz“. Die Sanduhr zeigt Ihnen, daß EXCEL arbeitet. Nach einigen Sekunden verschwindet sie wieder. Ist Ihnen aufgefallen, daß der Bildschirm einige Male geflackert hat?

Öffnen Sie jetzt das Dialogfeld „Seite einrichten“ wieder. Im Register „Tabelle“ ist das Kontrollkästchen „Gitternetzlinien“ deaktiviert, das Kontrollkästen „Schwarzweißdruck“ aktiviert. Der Makro macht genau das, was wir wollten!

Allerdings hat er einige Schönheitsfehler. Und die wollen wir jetzt herausfinden und beseitigen.

© by Josef Broukal, Vienna, Austria

Öffnen Sie zunächst über das Menü „Fenster“ die Arbeitsmappe, die EXCEL für Ihren ersten Makro angelegt hat. Das Blatt, das den Makro enthält, heißt „Modul1“. Was hat EXCEL da aufgeschrieben? 34 Zeilen, und nur in zweien davon steht das, was Sie festgehalten wissen wollten. Diese beiden Zeilen habe ich rot eingefärbt, damit Sie sie leichter finden:

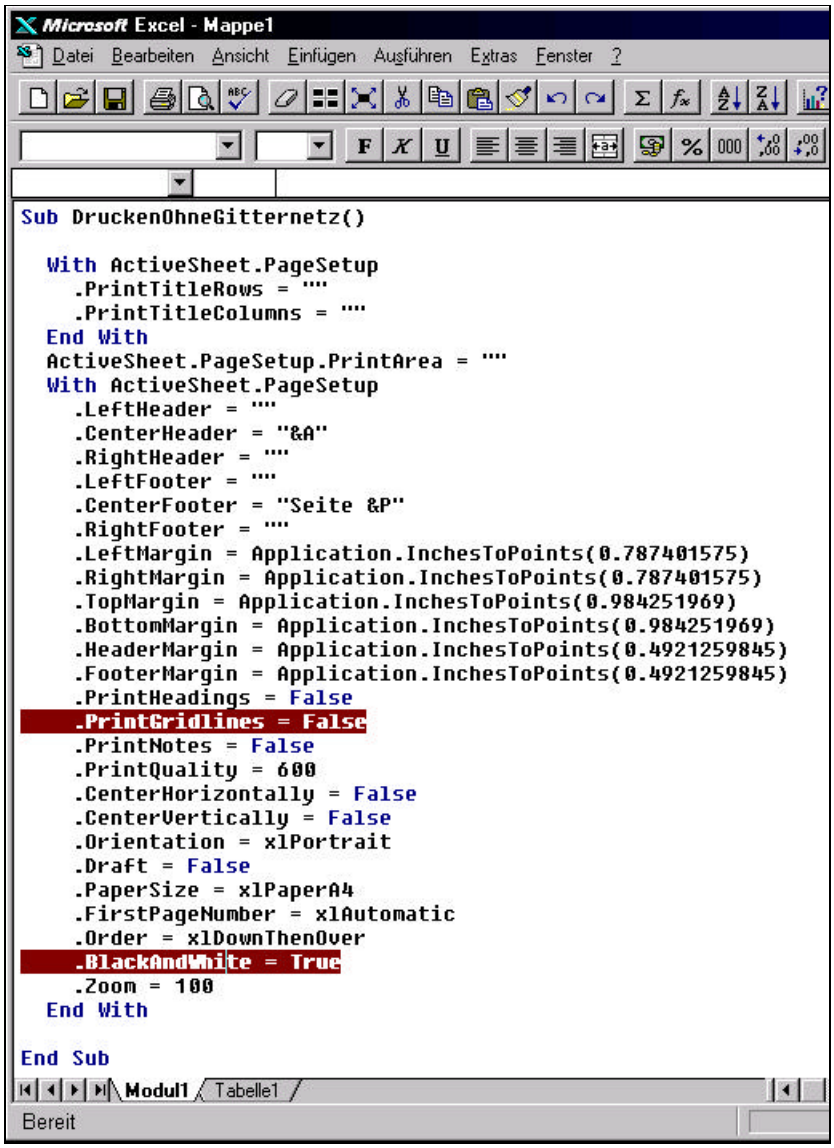


Abbildung 4: Der Makro-Rekorder protokolliert unsere Anweisung, eine Tabelle ohne Gitternetzlinien und mit Umwandlung von Farben in Schwarz/Weiß zu drucken. Leider schreibt er auch sehr viel Unnötiges mit—das müssen wir wieder entfernen!

Was lernen wir daraus? Wenn der Makro–Rekorder läuft, während Sie ein Dialogfeld bearbeiten, schreibt EXCEL alle Einstellungen auf, die in diesem Dialogfeld vorgenommen werden. Das hat unangenehme Folgen:

- Der Makro braucht lange, weil er viele Dinge unnötig setzt (deshalb das Flackern und die Sanduhr).
- Der Makro hält Dinge fest, die vielleicht nur für den Computer gelten, auf dem Sie den Makro erstellen. „Druckqualität = 600“ ist zum Beispiel eine Option, die beim HP–Laserjet 4 Sinn macht. Schon beim LaserJet 3—und erst recht bei Druckern anderer Hersteller—führt sie dazu, daß das Makroprogramm unterbrochen wird und EXCEL eine Fehlermeldung anzeigt.
- Verschiedene andere Einstellungen, vom Kopfzeilen–Text bis zum Papierrand und zum Papierformat, werden ebenfalls gesetzt, wann immer Sie den Makro laufen lassen.

© by Josef Broukal, Vienna, Austria

Was tun? Die Lösung ist einfach. Wir müssen:

- zur Arbeitsmappe wechseln, in der unser VBA–Modul steht
- das Modulblatt auswählen
- allen überflüssigen Code löschen

5. Nachbearbeiten

Zäumen wir das Pferd von hinten auf. Nach dem Löschen des überflüssigen Codes sollen folgende Codezeilen übrig bleiben:

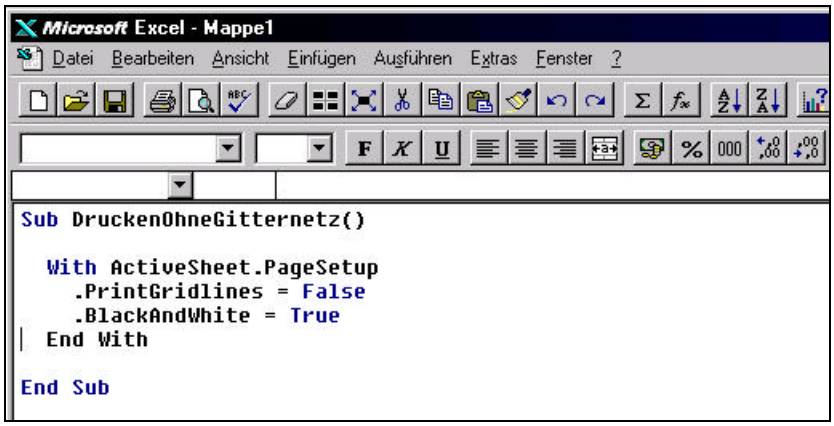


Abbildung 5: Der „geputzte“ Makro enthält nur noch jene Anweisungen, die wir wirklich geben wollten!

Den Rest löschen Sie nach Markieren mit der Maus oder per Tastatur (UMSCHALT+PFEIL–AB markiert jeweils eine ganze Zeile) durch Drücken der ENTF–Taste.

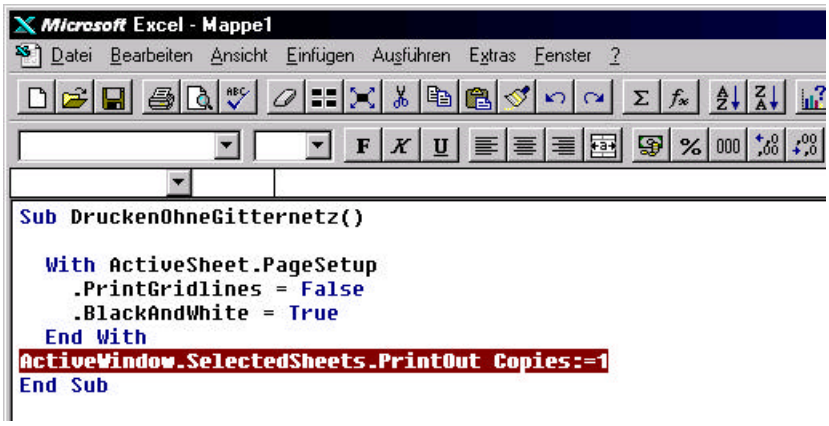
Die Aufzeichnung nachbessern

Eines fehlt unserem kleinen Programm noch: es führt nicht dazu, daß EXCEL auch tatsächlich druckt. Das wollen wir jetzt nachholen und dabei den Makro–Rekorder in seiner zweiten Arbeitsweise kennenlernen: Dem Aufzeichnen einzelner Befehle mitten in bereits fertigen Code hinein. Folgende Arbeitsschritte sind dazu notwendig:

- Setzen Sie den Cursor an den Beginn der letzten Zeile des Makros „DruckenOhneGitternetz“, also vor „End Sub“.
- Wählen Sie über das Menü „Extras“ und den Befehl „Makro aufzeichnen“ den Befehl „Position festlegen“. EXCEL wird an dieser Stelle mit dem Aufzeichnen beginnen.
- Wechseln Sie in eine Tabelle.
- Geben Sie über das Menü „Extras“ und den Befehl „Makro aufzeichnen…“ den Befehl „Ab Position aufzeichnen“. In der Statuszeile informiert Sie die Schrift „Aufzeichnung“ darüber, daß EXCEL ab sofort wieder „mitschreibt“.
- Wählen Sie im Menü „Datei“ den Befehl „Drucken…“. Alle Einstellungen des Dialogfeldes passen; schliessen Sie daher mit ENTER oder Mausklick auf die OK–Schaltfläche die Arbeit mit dem Dialogfeld ab
- Stoppen Sie über das Menü „Extras“, „Makro aufzeichnen…“ und „Aufzeichnung beenden“ den Makro–Rekorder.

Wechseln Sie bitte wieder in das Modulblatt. Der Makro–Rekorder hat eine neue Codezeile eingefügt:

© by Josef Broukal, Vienna, Austria



Der Eintrag „Copies:=1“ legt uns wieder unnötig Fesseln an. Löschen Sie diesen Text.

Damit Sie nicht zu fröhlich werden...

...sollten Sie noch wissen, was alles fehlt, bevor dieses Programm „deppensicher“ ist! Es sind im Wesentlichen zwei Dinge:

- Das Programm prüft nicht, ob überhaupt eine Arbeitsmappe geladen ist. Nur dann gibt es ein „aktives Blatt“!
- Selbst wenn es ein „aktives Blatt“ gibt: Wer sagt, daß es ein Tabellenblatt ist? Es könnte genauso gut ein VBA-Modul, ein Dialogblatt oder ein Diagrammblatt sein. Auch das müssen Sie abprüfen, bevor Sie diesen Code auf die Menschheit loslassen.

Nach Einbau dieser Verbesserungen sieht der Code dann so aus:

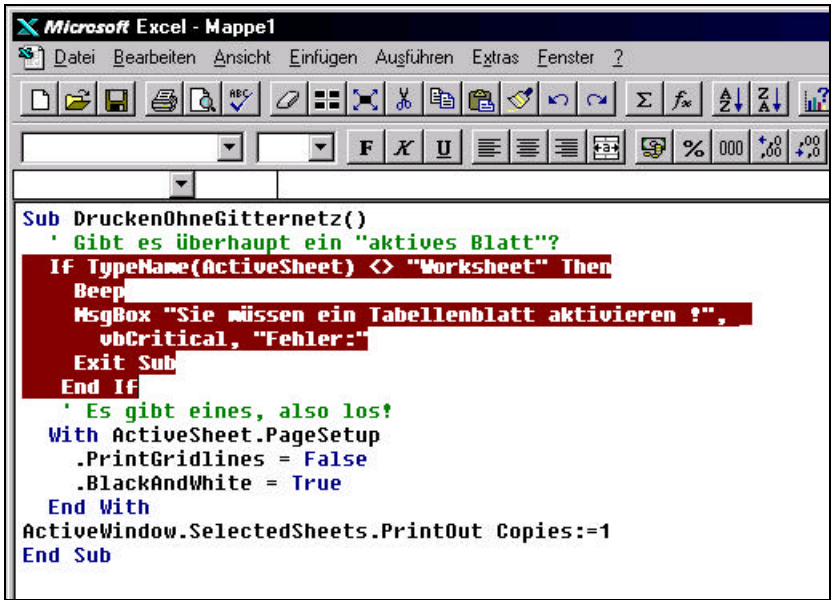


Abbildung 6: So prüfen Sie auf einen Schlag ab, ob es ein aktives Blatt gibt und ob dieses vom Typ „WorkSheet“ ist!

© by Josef Broukal, Vienna, Austria

So starten Sie VBA–Programme

Wir haben weiter oben einen der Wege kennengelernt, ein VBA–Programm zu starten: Über einen Befehl im Menü „Extra“. EXCEL gibt uns aber noch mehr Möglichkeiten an die Hand. Hier ist eine Übersicht.

- **Befehl im Menü „Extras“**. Eintrag entweder gleich beim Anlegen des Makros im Dialog „Makro aufzeichnen…“ über die Schaltfläche „Optionen“ oder später bei sichtbarem Modulblatt über „Extras“, „Makro…“, (Makronamen einmal anklicken),„Optionen“.
- **Befehl in einem anderen Menü.**
- **Befehl in einem selbstgeschaffenen Menü** einer eingebauten Menüleiste..
- **Befehl in einem Menü in einer selbstgeschaffenen Menüleiste**.
- **Tastenkombination von STRG und einer Buchstabentaste**, ausgenommen die Umlaute. EXCEL unterscheidet zwischen großen und kleinen Buchstaben.
- **Verbinden des Makros mit einem Symbol**. Sie können ein bestehendes Symbol verwenden oder aus der Abteilung „Benutzerdefiniert“ des Dialoges „Symbolleisten…“ (Menü „Ansicht“) eines auswählen. Ist der Dialog „Benutzerdefiniert“ sichtbar, können Symbole in einer Symbolleiste abgelegt werden; sie werden dann zu einem Teil dieser Symbolleiste. Ist der Vorgang beendet, öffnet EXCEL den Dialog „Zuweisen“ und bietet Ihnen die Namen aller geladenen VBA–Programm zur Auswahl an. Legen Sie das Symbol außerhalb einer bestehenden Symbolleiste ab, legt EXCEL eine neue Symbolleiste an.
- **Schaltknopf auf Tabellenblättern**.
- **Aufruf von einem anderen VBA–Programm aus**.
- **Beim Laden einer Arbeitsmappe.**
- **Beim Aktivieren eines Blattes.**
- **Beim Deaktivieren eines Blattes.**
- **Beim Schließen einer Arbeitsmappe.**

© by Josef Broukal, Vienna, Austria

VBA-Programme formal gesehen

Jedes VBA-Programm besteht zumindest aus einer „Prozedur“ oder „Funktion“. Eine „Prozedur“ ist die kleinste selbständige Code-Einheit, die Sie starten und ablaufen lassen können.

Prozeduren und Funktionen werden in „Modulblättern“ geschrieben und gespeichert. Jedes Modulblatt kann eine praktisch beliebige Anzahl von Prozeduren und Funktionen aufnehmen.

Eine Prozedur beginnt immer mit einer Codezeile, in der zunächst das Wort „Sub“ steht, gefolgt vom Namen der Prozedur:

```
Sub SetPercentInStatusBar(Percentage As Integer)
```

Eine Prozedur endet immer mit einer Codezeile, in der die Anweisung „End Sub“ steht; eine Funktion mit der Codezeile „End Function“.

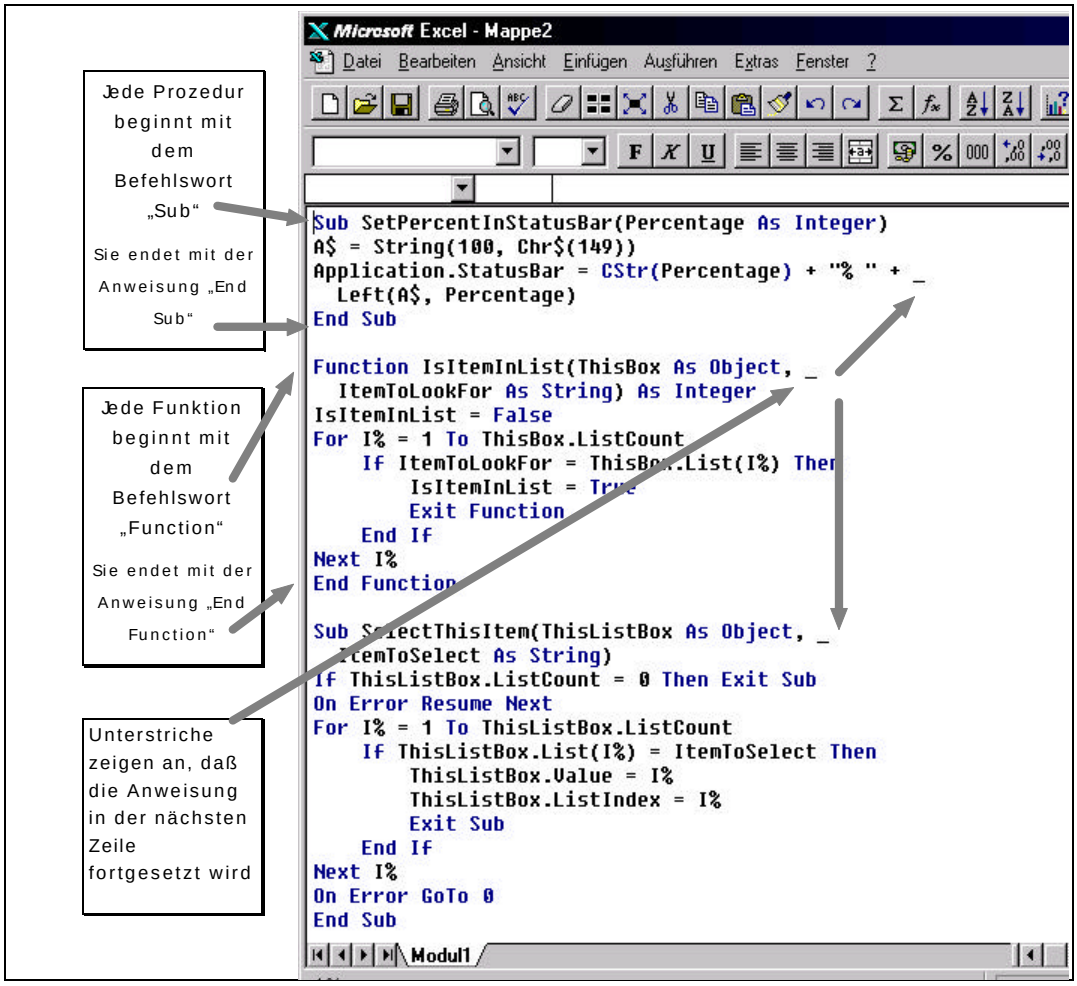

Jede Prozedur beginnt mit dem Befehlswort „Sub“
Sie endet mit der Anweisung „End Sub“
Jede Funktion beginnt mit dem Befehlswort „Function“
Sie endet mit der Anweisung „End Function“
Unterstriche zeigen an, daß die Anweisung in der nächsten Zeile fortgesetzt wird
Microsoft Excel - Mappe2
Datei Bearbeiten Ansicht Einfügen Ausführen Extras Fenster ?
Sub SetPercentInStatusBar(Percentage As Integer)
A$ = String(100, Chr$(149))
Application.StatusBar = CStr(Percentage) + "% " + _
Left(A$, Percentage)
End Sub
Function IsItemInList(ThisBox As Object, _
ItemToLookFor As String) As Integer
IsItemInList = False
For I% = 1 To ThisBox.ListCount
If ItemToLookFor = ThisBox.List(I%) Then
IsItemInList = True
Exit Function
End If
Next I%
End Function
Sub SelectThisItem(ThisListBox As Object, _
ItemToSelect As String)
If ThisListBox.ListCount = 0 Then Exit Sub
On Error Resume Next
For I% = 1 To ThisListBox.ListCount
If ThisListBox.List(I%) = ItemToSelect Then
ThisListBox.Value = I%
ThisListBox.ListIndex = I%
Exit Sub
End If
Next I%
On Error GoTo 0
End Sub
Modul1

Abbildung 7: Zwei Prozeduren, eine Funktion in einem EXCEL-„Modulblatt“. Zwischen Kopf und Fuß der Prozeduren und Funktionen werden die „Anweisungen“ eingeschlossen.

Zwischen diesen beiden Zeilen formulieren Sie eine beliebige Anzahl von „Anweisungen“, also Befehlen an EXCEL. Jede Anweisung beginnt in einer neuen Zeile.

Anweisung auf mehrere Zeilen aufteilen Ist der Text einer Anweisung so lang, daß er nicht mehr auf einen Blick auf dem Bildschirm zu sehen ist, sollte er auf mehrere Zeilen aufgeteilt werden. Damit

© by Josef Broukal, Vienna, Austria

VBA zum Beispiel weiß, daß ein Befehl in einer zweiten Codezeile fortgesetzt wird, müssen Sie am Ende der ersten Codezeile hinter dem Buchstaben zuerst einen Leerschritt einfügen, dann einen Unterstrich („_“). In der zweiten und den folgenden Zeilen gilt das Gleiche:

```
Worksheets("Tabelle1").Range("A1:C20").Sort _
  key1:=Worksheets("Tabelle1").Range("A1"), _
  key2:=Worksheets("Tabelle1").Range("B1")
```

Hinweis Codezeilen außerhalb von Prozeduren oder Funktionen können nicht ausgeführt werden!

© by Josef Broukal, Vienna, Austria

Nützen Sie die Online-Hilfe!

VBA ist eine sehr schöne Computer-Sprache. Es erklärt sich praktisch von selbst und verwendet sprechende Befehlswörter. Weil es davon (und von den Argumenten der Befehlswörter) aber viele hundert gibt, merkt man sich nur die häufig verwendeten. Zum Glück arbeitet VBA sehr eng mit der sehr ausführlichen Online-Hilfe zusammen.

Beispiel Angenommen, Sie wissen nicht mehr, welche Befehlswörter der Befehl zum Sortieren eines Tabellenbereiches braucht. Schreiben Sie einfach das Wort „sort“ in ein Modulblatt. Drücken Sie dann die Funktionstaste „F1“. EXCEL zeigt Ihnen den Hilfetext für „Sort“. Wann immer Sie den Cursor in ein Befehlswort stellen und dann „F1“ drücken, sucht EXCEL die zugehörige Hilfeseite. Kann es sie nicht finden (z.B. weil Sie im Befehlswort einen Schreibfehler gemacht haben), erscheint die Seite „Schlüsselwort nicht gefunden“.

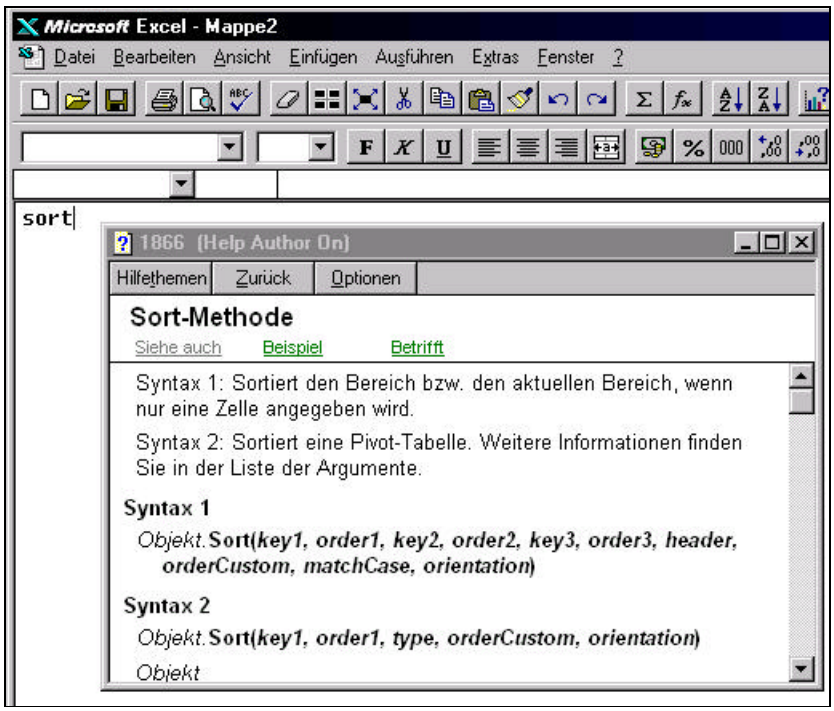

Microsoft Excel - Mappe2
Datei Bearbeiten Ansicht Einfügen Ausführen Extras Fenster ?
sort
1866 (Help Author On)
Hilfethemen Zurück Optionen
Sort-Methode
Siehe auch Beispiel Betrifft
Syntax 1: Sortiert den Bereich bzw. den aktuellen Bereich, wenn nur eine Zelle angegeben wird.
Syntax 2: Sortiert eine Pivot-Tabelle. Weitere Informationen finden Sie in der Liste der Argumente.
Syntax 1
Objekt.Sort(key1, order1, key2, order2, key3, order3, header, orderCustom, matchCase, orientation)
Syntax 2
Objekt.Sort(key1, order1, type, orderCustom, orientation)
Objekt

Abbildung 8: Drücken von „F1“ zeigt die Online-Hilfe zu dem Befehlswort, in oder hinter dem der Cursor steht.

Markieren Sie mit der Maus den beim Punkt „Syntax“ der Hilfe angegebenen Befehlswortlaut mit der Maus und drücken dann „Strg“+“C“, so kopiert Windows das Markierte in die Zwischenablage. Durch Einfügen in ein Modulblatt erhalten Sie eine „leere“ Anweisung. Ihren Argumenten brauchen Sie nur noch die Werte zuzuweisen. Unmittelbar nach dem Kopieren zeigt EXCEL an, daß die Anweisung nicht in Ordnung ist. Stimmt! Und zwar so lange, bis alle Argumente ihre Werte gewiesen bekommen haben oder gelöscht wurden (wo das Sinn macht).

Hinweis Wenn Sie EXCEL oder Microsoft Office für Windows 95 mit der Option „Standard“ installiert haben, wird die VBA-Hilfe nicht auf Ihren Computer übertragen. Wenn Sie das wollen, müssen Sie das Setup-Programm nochmals laufen lassen und dabei die Option „Benutzerdefiniert“ wählen. Jetzt haben Sie die Möglichkeit, auch das Installieren der VBA-Hilfe anzuordnen.

© by Josef Broukal, Vienna, Austria

Online-Hilfe ergänzen

Falls Sie einmal zu einem Befehl etwas Merkenswertes entdecken, das in der Online-Hilfe fehlt, können Sie eine „Anmerkung“ auf der Hilfeseite einfügen. Klicken Sie dazu mit der Maus auf die Schaltfläche „Optionen“. Wählen Sie dann im Menü den Befehl „Anmerken...“ aus. Bereits vorhandene Anmerkungen zeigt Ihnen Windows durch eine grüne Heftklammer unterhalb der Überschrift der Hilfeseite.

© by Josef Broukal, Vienna, Austria

Die Symbolleiste „Visual Basic“

In diesem Kapitel lernen Sie:

- Elemente der Symbolleiste „Visual Basic“
- Programm schrittweise abarbeiten
- Haltepunkt setzen
- Direktbereich benützen
- Code im Direktbereich testen
- Überwachungsausdrücke hinzufügen

Wir haben unseren ersten Makro, „DruckenOhneGitternetz“, mit dem Makro–Rekorder erstellt und über das Menü „Extras“ zum Laufen gebracht. Das genügt bei einem so einfachen Programm. Bevor wir aber ans wirkliche VBA–Programmieren gehen, wollen wir die Werkzeuge kennenlernen, die EXCEL uns zur Verfügung stellt. Sie sind in der Symbolleiste „Visual Basic“ enthalten.

Elemente der Symbolleiste „Visual Basic“

Wechseln Sie bitte wieder in das VBA–Modulblatt, in dem EXCEL das Programm „DruckenOhneGitternetz“ aufgezeichnet hat. Sicher fällt Ihnen auf, daß EXCEL in diesem Augenblick eine neue Symbolleiste anbietet. Sie sieht so aus:

Abbildung 9: Die Symbolleiste „Visual Basic“, mit deren Hilfe Sie VBA–Programme aufzeichnen, starten und testen können.

Die einzelnen Symbole haben folgende Bedeutung:

Neues Visual Basic–Modul Fügt ein neues Visual Basic-Modul in die aktive Arbeitsmappe ein.

Menü–Editor Zeigt das Dialogfeld Menü-Editor an, mit dem Sie Microsoft Excel-Menüs *für die aktive Arbeitsmappe* erstellen und bearbeiten können.

Objekt–Katalog Der Objekt–Katalog zeigt Ihnen die Namen aller Prozeduren, Funktionen, Methoden, Eigenschaften und Objekte aus allen geöffneten Arbeitsmappen. Sie können entweder zum Code springen oder den Code, mit dem Prozedur oder Funktion angesprochen werden, an die Cursor–Position in Ihrem aktiven Modulblatt kopieren.

Makro ausführen Führt den Makro aus, in dem der Cursor steht; steht der Cursor nicht in einem Makro, wird das Dialogfeld Makro angezeigt, in dem Sie einen Makro auswählen und ausführen können.

Makro-Einzelschritt Führt nur die erste Zeile des Programmes aus; zeigt dann das „Testfenster“.

Makro fortsetzen Nimmt das Ausführen eines angehaltenen Makros wieder auf; beendet Modus „Einzelschritt“ und führt den Makro ohne Unterbrechnung weiter aus.

Makro–Aufzeichnung beenden Beendet das Ausführen des gerade laufenden Makros. Beendet das Aufzeichnen mit dem Makro–Rekorder.

Makro–Aufzeichnung beginnen Zeigt das Dialogfeld „Makro aufzeichnen“ an und startet dann den Makro–Rekorder.

Haltepunkte ein-/ausschalten Hält ein VBA–Programm an und wechselt in den Unterbrechungsmodus.

© by Josef Broukal, Vienna, Austria

Aktuellen Wert zeigen Zeigt im Dialogfeld „Aktuellen Wert anzeigen“ den augenblicklichen Wert des gewählten (markierten) Ausdruckes.

Einzelschritt Führt eine Zeile eines VBA–Codes aus und kehrt dann wieder in den Unterbrechungsmodus zurück.

Prozedurschritt Führt eine Zeile eines VBA–Codes aus und kehrt dann in den Unterbrechungsmodus zurück. Wird in dieser Zeile ein Unterprogramm aufgerufen, so wird dieses ohne Unterbrechnung abgearbeitet.

Wir wollen als nächstes die Arbeitsweise dieser Schaltknöpfe demonstrieren. Legen Sie dazu zunächst ein neues Modulblatt an.

Schreiben Sie dann den folgenden Code:

```
Sub Hauptprozedur()

Wert1 = 5
Wert2 = 12
Produkt = Wert1 * Wert2

End Sub
```

Setzen Sie in der Zeile „End Sub“ einen Haltepunkt.

Klicken Sie auf den Schaltknopf „Makro ausführen“.

Sobald das VBA–Programm die Zeile mit dem Haltepunkt erreicht hat, geht es in den Unterbrechnungs–Modus und EXCEL zeigt Ihnen das „Testfenster“:

Das VBA–Testfenster

Im Testfenster können Sie einzelne Codezeilen ausführen, Variablenwerte anzeigen, Überwachungsausdrücke ansehen und weitere Tests durchführen.

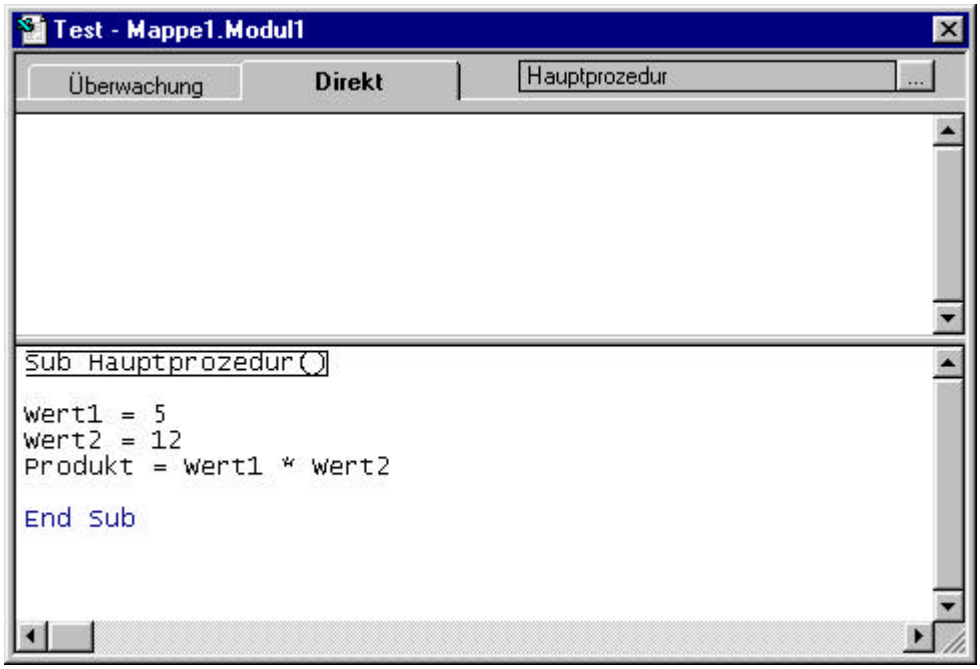


Abbildung 10: Das VBA–Testfenster.

© by Josef Broukal, Vienna, Austria

Sie zeigen das Testfenster an, indem Sie aus dem Menü „Ansicht“ den Befehl „Testfenster“ wählen.

Das Testfenster wird geöffnet

- wenn ein Haltepunkt erreicht wird
- wenn Sie manuell in den Unterbrechungsmodus wechseln. Dies können Sie immer dann tun, wenn ein VBA-Programm wegen eines Fehlers anhält.

Hinweis Während der Ausführung können keine Anweisungen im Direktbereich des Testfensters eingegeben werden. Mit dem Debug-Objekt (so heißt das Testfenster auf Englisch) und der Print-Methode können Sie jedoch Werte aus dem gerade laufenden Program ins Direktfenster hinschreiben und sie ansehen, sobald das Programm wieder anhält. Der folgende Code gibt zum Beispiel den Wert der Variablen A im Testfenster aus:

```
Debug.Print A
```

und der folgende Code gibt noch erklärenden Text dazu:

```
Debug.Print “Wert von A in Prozedur X“; A
```

Der Strichpunkt zwischen Text und Variablenname („A“) führt dazu, daß A mit nur einem Zeichen Abstand zum Text ausgedruckt wird. Ein Beistrich als Trenner läßt einige Zeichen Abstand.

Wenn ein Programm unterbrochen wurde, können Sie auch Anweisungen im Direktbereich des Testfensters eingeben und dort ausführen lassen.

Das Testfenster besteht aus folgenden Elementen:

Direktbereich Im Direktbereich wird Code zur sofortigen Ausführung eingegeben. Er wird standardmäßig beim ersten Öffnen des Testfensters angezeigt. Wählen Sie das Register Direkt zur Anzeige des Direktbereichs, wenn der Überwachungsbereich angezeigt wird.

Überwachungsbereich Der Überwachungsbereich zeigt Informationen zu allen definierten Überwachungsausdrücken an. Wählen Sie das Register Überwachung zur Anzeige des Überwachungsbereichs.

Prozedurfeld Das Prozedurfeld zeigt im Unterbrechungsmodus den Namen des Projekts und des Moduls an, und den Namen der Prozedur, in der die Ausführung unterbrochen wurde. Mit der Schaltfläche rechts neben dem Prozedurfeld kann das Dialogfeld Aufrufe angezeigt werden. Es zeigt Ihnen, welche Programmteile bis zu der Codezeile ausgeführt wurden, in der Ihr Programm angehalten hat.

Bildschirmteiler Der Bildschirmteiler unterteilt das Testfenster in zwei Bereiche. Der obere Bereich stellt den Direkt- oder Überwachungsbereich dar. Der Codebereich unterhalb des Bildschirmteilers zeigt den Code an der Stelle an, an der die Ausführung unterbrochen wurde. Sie ändern die Größe eines Bereichs, indem Sie den Bildschirmteiler nach oben oder nach unten ziehen.

Codebereich Der Codebereich zeigt den Prozedurcode an der Stelle, an der die Ausführung unterbrochen wurde.

Programme schrittweise abarbeiten

Wir wollen zunächst anhand des kurzen VBA–Programmes auf Seite 15 die Arbeitsmöglichkeiten mit der Makro–Symbolleiste erkunden.

© by Josef Broukal, Vienna, Austria

Klicken Sie auf den Schaltknopf „Makroeinzelschritt“. VBA zeigt Ihnen das Testfenster. Die erste Zeile des Codes ist eingerahmt. Dies ist die Zeile, die VBA gerade abgearbeitet hat.

Drücken Sie auf den Schaltknopf „Einzelschritt“. VBA arbeitet die zweite Zeile ab; jetzt ist diese Zeile eingerahmt.

Drücken Sie nochmals auf den Schaltknopf „Einzelschritt“. VBA arbeitet die dritte Zeile ab.

Lassen sie das VBA–Programm jetzt ohne Unterbrechnung bis zum Schluß ablaufen. Drücken Sie dazu den Schaltknopf „Makro fortsetzen“. Das Testfenster verschwindet.

Haltepunkt setzen

Setzen Sie den Cursor in die Zeile mit dem Code

```
Wert2 = 12
```

Drücken Sie den Schaltknopf „Haltepunkt ein-/ausschalten“.

Drücken Sie den Schaltknopf „Makro ausführen“

VBA kennzeichnet diese Zeile, in dem es einen (standardmäßig roten) Balken unter die Zeile legt und den Codetext weiß darstellt:

```
Wert2 = 12
```

Der Makro läuft bis in die mit dem Haltepunkt versehene Zeile; dann zeigt EXCEL das Testfenster. Jetzt können Sie im Fenster „Direkt“ Ihren Code testen oder VBA andere Aufgaben lösen lassen.

Direktbereich benützen

Variablenwerte erfragen und überwachen

Setzen Sie den Cursor in das Fenster „Direkt“. Geben Sie den folgenden Code ein:

```
? Wert1
```

Drücken Sie die Eingabetaste. VBA gibt in der nächsten Zeile des Direktfensters das Ergebnis der Berechnung aus:

```
? Wert1
 5
```

Wie Sie sehen, entspricht das Fragezeichen dem Befehl „Gib den Wert der Variablen rechts vom Fragezeichen aus“.

Variablenwerte neu bestimmen

Setzen Sie den Cursor in die erste freie Zeile des Direktfensters. Geben Sie ein:

```
Wert1=8
```

Drücken Sie die Eingabetaste. Schreiben Sie in die nächste Zeile:

© by Josef Broukal, Vienna, Austria

```
? Wert1
```

EXCEL antwortet mit:

```
 8
```

und hat somit diesen Wert zur Kenntnis genommen.

Drücken Sie jetzt solange den Schaltknopf „Einzelschritt“, bis die Zeile „Ende Sub“ zur aktiven Zeile im VBA–Programm wird.

Wechseln Sie wieder ins Direktfenster. Wenn Sie VBA die Frage nach dem Wert des Produkts von „Wert1“ und „Wert2“ stellen, erhalten Sie als Antwort „96“. VBA hat also mit dem von Ihnen händisch veränderten Wert weitergearbeitet.

```
Wert1=8
? Wert1
 8
? Produkt
 96
```

Den aktuellen Wert eines Ausdruckes zeigen

Wenn Sie die Codezeile, aus der Sie einen Wert erfragen wollen, im Codebereich sehen, können Sie auch so vorgehen:

Setzen Sie den Cursor in das Codewort „Wert1“.

Markieren sie das ganze Wort durch Maus–Doppelklick.

Klicken Sie auf den Schaltknopf „Aktuellen Wert zeigen“ oder wählen Sie im Menü „Extras“ den Befehl „Aktuellen Wert anzeigen“ oder drücken Sie UMSCHALT+F9.

VBA zeigt Ihnen darauf das Dialogfeld „Aktuellen Wert anzeigen“:

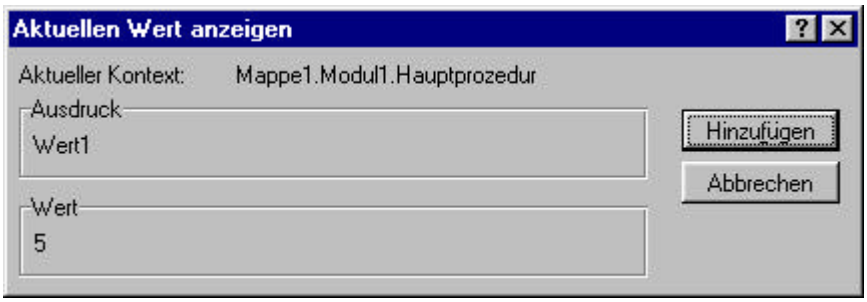


Abbildung 11: In diesem Dialogfeld zeigt VBA den aktuellen Wert eines von Ihnen markierten Code–Ausdruckes.

Das Dialogfenster „Aktuellen Wert anzeigen“

Der Dialog „Aktuellen Wert anzeigen“ zeigt den letzten Wert eines ausgewählten Ausdrucks an. Erhält eine Variable während des Programmlaufes mehrere Werte, so wird doch, egal in welcher Zeile sie den Ausdruck auswählen, immer nur der letzte Wert angezeigt.

© by Josef Broukal, Vienna, Austria

Die Elemente des Dialogs „Aktuellen Wert anzeigen“ sind:

Schaltfläche „Hinzufügen“ Fügt den Ausdruck zum Überwachungsbereich hinzu.

Feld "Aktueller Kontext" Die Namen der Arbeitsmappe, des Moduls und der Prozedur, in der sich der Überwachungsausdruck befindet.

Feld "Ausdruck" Der ausgewählte Ausdruck.

Feld "Wert" Der Wert des ausgewählten Ausdrucks. Der aktuelle Wert wird nicht angezeigt, wenn der Kontext des Ausdrucks außerhalb der Prozeduren liegt, die im Dialog „Aufrufe“ aufgeführt sind.

Code im Direktbereich testen

Wir haben im Direktbereich weiter oben den Wert einer bestehenden Variablen erfragt und später verändert. Sie können dort aber weitaus mehr tun:

Im Direktbereich eingegebener Code wird sofort ausgeführt. Sie können eine Anweisung in den Direktbereich eingeben oder einfügen und sie durch Drücken der EINGABETASTE sofort ausführen. Der folgende Code zeigt zum Beispiel den Wert der Variablen A an, in diesem Fall 5.

```
?A
 5
```

Sie können aber auch neue Variable schaffen und ihnen Werte zuweisen:

```
NeueVariable=7
? NeueVariable
  7
```

Zwei Betriebszustände des Direktfensters sind möglich: „Wartemodus“ und „Unterbrechungsmodus“. „Wartemodus“ ist der Zustand nach den Beenden eines VBA–Programmes. Das Testfenster ist über das Menü „Ansicht“ erreichbar, sobald das aktuelle Blatt ein Modulblatt ist. Im „Unterbrechungsmodus“ ist ein VBA–Programm dann, wenn es an einem Haltepunkt angelangt ist oder an einer „Stop“–Anweisung, an einer fehlerhaften Zeile oder an einem überwachten Ausdruck, bei dem Sie festgelegt haben, daß die Programmausführung unterbrochen werden soll.

Im Wartemodus wird der Code in einem globalen Kontext ausgeführt. Im Unterbrechungsmodus wird der Code im Direktbereich in dem Kontext ausgeführt, der durch die Titelleiste des Testfensters angegeben wird. Wenn Sie zum Beispiel „print <Variablenname>“ eingeben, ist die Ausgabe der Wert einer lokalen Variablen. Dies ist dasselbe, als wenn die Print-Methode in der im Prozedurfeld angegebenen Prozedur ausgeführt worden wäre.

Der Direktbereich erscheint standardmäßig beim ersten Öffnen des Testfensters. Durch Mausklick auf das Register „Direkt“ können Sie zum Direktbereich wechseln, wenn der Überwachungsbereich angezeigt wird.

Überwachungsausdrücke hinzufügen

Der Schaltknopf „Aktuellen Wert anzeigen“ gibt Ihnen auch die Möglichkeit, im Überwachungsbereich des Testfensters den Wert eines Ausdruckes während der Laufzeit eines Programmes permanent anzuzeigen und zu untersuchen. Wählen Sie dazu den Schaltknopf „Hinzufügen“.

Wählen Sie dazu im Testfenster die Registermarke „Überwachung“.

© by Josef Broukal, Vienna, Austria

Wenn der Wert nicht verfügbar ist, wird eine der folgenden Meldungen angezeigt:

„Kann Modul nicht übersetzen“ Das Modul, in dem der Überwachungsausdruck definiert ist, enthält einen Syntaxfehler und kann nicht übersetzt werden, oder der Kontext des Überwachungsausdrucks ist mit "Alle Module" definiert, und eines der Module im Kontext kann nicht übersetzt werden.

„Ungültiger Überwachungsausdruck“ Der Überwachungsausdruck ist kein gültiger Ausdruck. Gültige Überwachungsausdrücke können Konstanten, Variablen, Funktion-Prozeduraufrufe sowie Elemente benutzerdefinierter Typen enthalten, jedoch keine Sub-Prozeduraufrufe oder benutzerdefinierte Typen.

„Außerhalb des Kontexts“ Der Wert des Überwachungsausdrucks ist innerhalb des momentan gültigen Kontexts nicht verfügbar. So sind beispielsweise im Wartemodus alle Überwachungsausdrücke außerhalb des Kontexts.

„Kann Datenfeld nicht überwachen“ Der Überwachungsausdruck ist ein Datenfeld. Es können jedoch nur einzelne Elemente von Datenfeldern überwacht werden.

Fügen Sie aus unserem kleinen Codestück alle Variablennamen: „Wert1“, „Wert2“ und „Produkt“ in den Überwachungsbereich ein. Lassen Sie dann den Code vom Beginn weg im Einzelschritt–Modus laufen. Beobachten Sie, was sich im Überwachungsfenster tut.

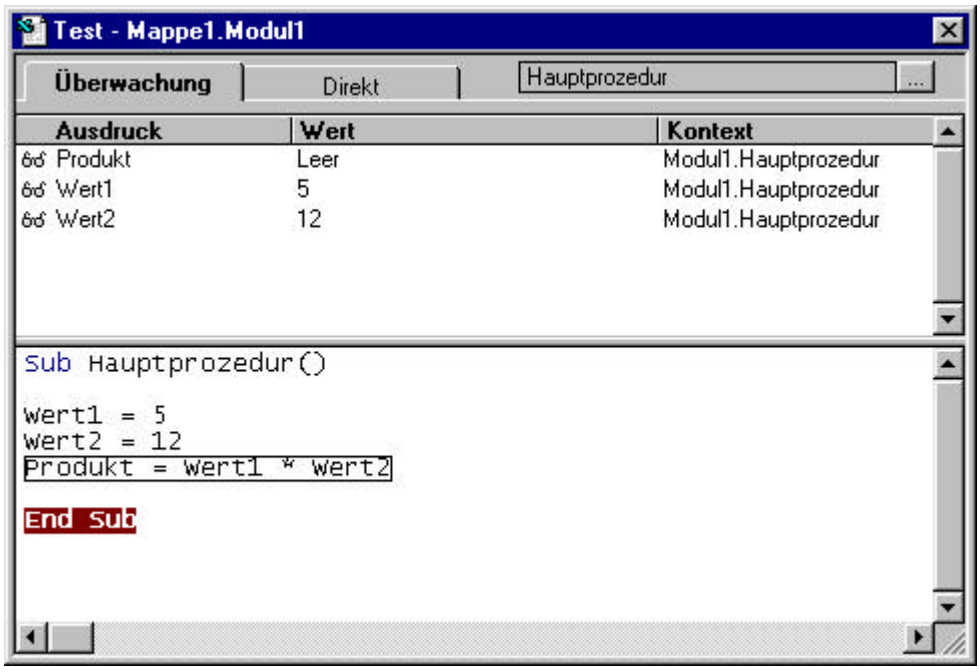


Abbildung 12: **Im Überwachungsbereich des Testfensters finden Sie das Ergebnis der überwachten Ausdrücke. Der Ausdruck „Produkt“ ist noch nicht ausgewertet worden und daher „Leer“.**

Mehr Möglichkeiten, einen Ausdruck zu überwachen, erhalten Sie über das Menü „Extras“ und den Befehl „Überwachung hinzufügen…“. Wenn Sie diesen Befehl wählen, zeigt EXCEL das Dialogfeld „Überwachung hinzufügen“:

© by Josef Broukal, Vienna, Austria

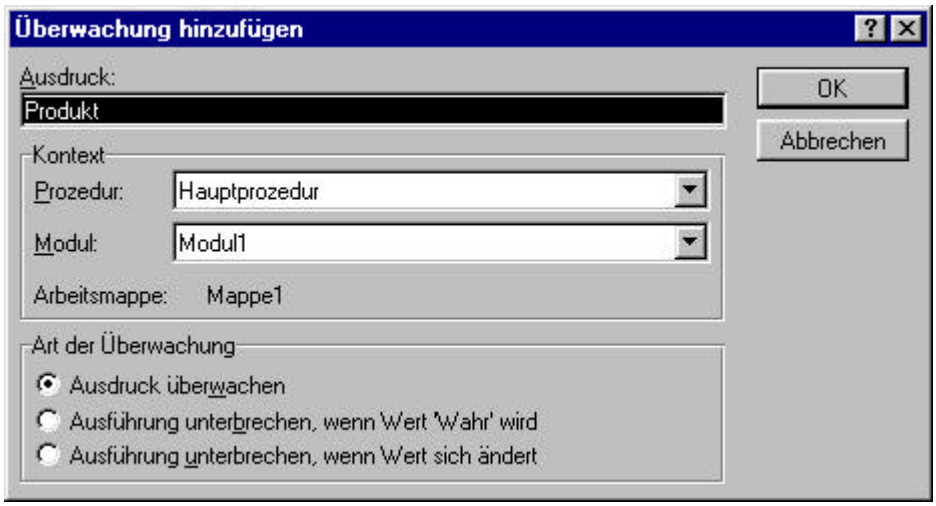


Abbildung 13: **Dieses Dialogfeld läßt Sie Art und Wirkungsweise einer Überwachung festlegen**

Der Dialog „Überwachung hinzufügen“ dient zur Eingabe eines Überwachungsausdrucks. Als Ausdruck kann jeder gültige Visual–Basic–Ausdruck verwendet werden. Überwachungsausdrücke werden im Überwachungsbereich des Testfensters aktualisiert, wenn Sie in den Unterbrechungsmodus wechseln oder nachdem Sie eine Anweisung im Direktbereich ausgeführt haben.

Wichtig Wählen Sie den kleinsten ausreichenden Gültigkeitsbereich. Wenn Sie alle Prozeduren oder Module auswählen, kann die Ausführungsgeschwindigkeit Ihres Programmes deutlich herabgesetzt werden, da der Ausdruck nach der Ausführung jeder einzelnen Anweisung ausgewertet wird. Wenn Sie eine bestimmte Prozedur als Kontext auswählen, wirkt sich das auf das Tempo nur so lange aus, wie die Prozedur in der Liste mit den aktiven Prozeduraufrufen aufgeführt ist. Sie können diese Liste anzeigen, indem Sie im Testfenster auf die Schaltfläche rechts neben dem Prozedurfeld klicken.

Und so gehen Sie beim Hinzufügen eines neuen Überwachungsausdruckes vor:

- Wählen Sie im Menü „Extras“ den Befehl „Überwachung hinzufügen“.
- Geben Sie einen Ausdruck im Feld "Ausdruck" ein. Wenn Sie im Programmcode bereits ein Ausdruck ausgewählt haben, wird dieser automatisch im Feld "Ausdruck" angezeigt:

  ⇒ Cursor im Variablennamen „Wert1“: VBA bietet „Wert1“ als zu überwachenden Ausdruck an.
  ⇒ Codeteil „Wert1 * Wert2“ markiert. VBA bietet „Wert1 * Wert2“ an.

- Sie fügen den Ausdruck im Feld "Ausdruck" hinzu, indem Sie "OK" wählen oder die EINGABETASTE drücken.

Die Elemente des Dialogs „Überwachung hinzufügen“ sind:

Feld "Ausdruck": Zeigt standardmäßig das Befehlswort im Code an, in dem der Cursor steht. Sie können aber auch einen anderen Ausdruck angeben.

Gruppe "Kontext":

Prozedur: Zeigt den Namen der Prozedur an, in der sich der ausgewählte Ausdruck befindet (Voreinstellung), und dient zum Definieren der Prozedur(en), in der/denen der Ausdruck ausgewertet wird. An dieser Stelle können Sie alle Prozeduren oder einen Prozedurkontext für die Auswertung des Ausdrucks angeben.

© by Josef Broukal, Vienna, Austria

Modul: Zeigt den Namen des Moduls an, in dem sich der ausgewählte Ausdruck befindet (Voreinstellung). An dieser Stelle können Sie alle Module oder einen bestimmten Modulkontext für die Auswertung des Ausdrucks angeben.

Projekt: Zeigt den Namen der aktuellen Arbeitsmappe an. Ausdrücke mit einem Kontext außerhalb der aktuellen Arbeitsmappe können nicht ausgewertet werden.

Gruppe "Art der Überwachung"

Ausdruck überwachen: Der Ausdruck und der zugehörige Wert werden im Überwachungsbereich des Testfensters angezeigt. Wenn Sie in den Unterbrechungsmodus wechseln, wird der Wert des Überwachungsausdrucks automatisch aktualisiert.

Ausführung unterbrechen, wenn Wert "Wahr" ist: Der Unterbrechungsmodus für das Modul wird automatisch aktiviert, sobald der Ausdruck innerhalb des angegebenen Kontexts den Wert Wahr ergibt.

Ausführung unterbrechen, wenn Wert sich ändert: Der Unterbrechungsmodus für das Modul wird automatisch aktiviert, sobald sich der Ausdruck innerhalb des angegebenen Kontexts ändert.

Weitere Möglichkeiten zur Arbeit mit dem Testfenster

Programm aus dem Testfenster heraus starten

Testfenster anzeigen (STRG+g). Aus dem Direkt-Bereich des Testfensters heraus die Prozedur oder Funktion aufrufen, die das Programm ins Laufen bringt.

Prozedur starten

Prozeduraufruf gefolgt von Parameter-Aufruf eintragen; dann ENTER-Taste drücken.

Funktion starten

Funktionsergebnis einer Variablen zuweisen oder mit ? zur Anzeige bringen; Parameter in Klammern setzen. Dann ENTER-Taste drücken.

Laufendes Programm anhalten und ins Testfenster wechseln

Im VBA-Code Fügen Sie dazu an der Stelle im VBA-Code, in der Sie Ihr Programm anhalten wollen, die Anweisung „Stop“ ein. EXCEL unterbricht seine Arbeit und zeigt Ihnen das Testfenster.

Per Tastatur Drücken Sie, während das Programm läuft, die Tasten „Strg“+“Untbr“. EXCEL unterbricht meistens (aber nicht immer!) die Ausführung und zeigt Ihnen das folgende Dialogfeld:

© by Josef Broukal, Vienna, Austria

Abbildung 14: **Auch wenn EXCEL glaubt, jede Unterbrechung eines Programmes durch Sie wäre ein „Makrofehler“ – dieses Dialogfeld gibt Ihnen mehrere Optionen, Ihren Code zu untersuchen**

Die Schaltflächen „Ende“, „Weiter“ und Hilfe erklären sich selbst. „Testen“ öffnet das Testfenster; „Gehe zu“ beendet die Code-Ausführung und bringt die Code-Zeile in den Vordergrund, bei deren Ausführung das Programm unterbrochen wurde.

Programm im Einzelschritt weiterlaufen lassen

F8 drücken.

Programm im Prozedurschritt weiterlaufen lassen

UMSCHALT+F8 drücken.

Haltepunkt setzen/löschen

Bringen Sie im Codebereich des Testfensters den Cursor in die Zeile, in der Sie den Haltepunkt setzen oder einen vorhandenen Haltepunkt löschen wollen. Drücken Sie dann F9.

Aktuellen Wert einer Variablen anzeigen

Bringen Sie-bei unterbrochenem Programm-im Codebereich des Testfenster den Cursor auf den Text der Variablen. Drücken Sie dann UMSCHALT+F9.

© by Josef Broukal, Vienna, Austria

Sprachbestandteile von VBA

In diesem Kapitel werden vorgestellt:

- Prozeduren
- Funktionen
- Anweisungen
- Variablen und Datentypen
- Konstanten
- Datenfelder
- Operatoren
- Objekte
- Gültigkeitsbereiche

Wir wollen in diesem Kapitel einen geordneten Überblick über das geben, was VBA ausmacht. Als Ausgangspunkt wollen wir wieder unser kleines VBA–Programm verwenden:

```
Sub Hauptprozedur()

Wert1=5
Wert2=12
Produkt=Wert1 * Wert2

End Sub
```

Was Sie hier sehen, heißt in VBA eine „Prozedur“.

Prozeduren

Eine Prozedur ist eine selbständige Einheit von Anweisungen, die („Argumente“) erhalten, eine Reihe von Anweisungen ausführen und die Werte der übergebenen Argumente ändern kann. Die Prozedur ist die kleinste selbständige Einheit eines VBA–Programmes. Sie beginnt mit dem Text

```
Sub
```

plus einem Namen. Dieser Name darf auf einem Modulblatt nur ein einziges Mal zum Bezeichnen einer Prozedur verwendet werden.

Am Ende jeder Prozedur steht die Zeile

```
End Sub
```

die VBA anzeigt, daß die Prozedur hier aufhört. In der nächsten Zeile kann eine weitere Prozedur beginnen.

Zwischen Beginn und Ende einer Prozedur befinden sich jene Codezeilen, in denen VBA wirklich „arbeitet“, die sogenannten „Anweisungen“.

© by Josef Broukal, Vienna, Austria

Syntax

```
[Private | Public][Static] Sub Name [(ArgListe)]
  [Anweisungen]
  [Exit Sub]
  [Anweisungen]
End Sub
```

Elemente

Die Sub-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Public	Auf die Sub-Prozedur kann von allen anderen Prozeduren in allen Modulen zugegriffen werden. Bei Verwendung in einem privaten Modul (mit einer Option Privat-Anweisung) kann auf die Prozedur nur innerhalb des Projekts zugegriffen werden.
Private	Auf die Sub-Prozedur kann nur durch andere Prozeduren aus dem Modulblatt zugegriffen werden, in dem steht.
Static	Die lokalen Variablen der Sub-Prozedur bleiben zwischen Aufrufen erhalten. Das Attribut Statisch wirkt sich nicht auf Variablen aus, die außerhalb der Sub-Prozedur deklariert wurden, auch wenn sie in der Prozedur verwendet werden.
Name	Name der Sub-Prozedur, gemäß den Standardkonventionen für Namen von Variablen.
ArgListe	Variablenliste mit den Argumenten, die an die Sub-Prozedur beim Aufruf übergeben werden. Mehrere Variablen werden durch Beistriche getrennt.
Anweisungen	Eine beliebige Gruppe auszuführender Anweisungen im Rumpf der Sub-Prozedur.

Die Syntax für das Argument ArgListe ist:

```
[[Optional][ByVal | ByRef][ParamArray] Variablenname[( )] As Typ]
```

Die ArgListe besteht aus folgenden Teilen:

Teil	Beschreibung
Optional	Ein Argument muß nicht angegeben werden. Alle im Anschluß an Optional in ArgListe angegebenen Argumente müssen auch optional sein und mit dem Schlüsselwort Optional deklariert werden. Alle als Optional angegebenen Argumente müssen den Typ Variant haben. Optional kann überhaupt nicht verwendet werden, wenn ArgumentListe verwendet wird. **Hinweis** In Ihrem Code können Sie mithilfe der Funktion IsMissing() überprüfend, ob ein optionales Argument übergeben worden ist.
ByVal	Das Argument wird als Wert übergeben.
ByRef	Das Argument wird als Referenz übergeben.
ParamArray	Ist nur als letztes Argument in ArgListe zulässig und gibt an, daß das letzte Element ein als Optional deklariertes Datenfeld mit Variant-Elementen ist. Das Schlüsselwort ParamArray erlaubt die Angabe einer variablen Anzahl von Argumenten und darf nicht in Kombination mit den Schlüsselwörtern ByVal, ByRef oder Optional verwendet werden. Ein Beispiel dafür auf Seite 27.
Variablenname	Name der Variablen, die das Argument repräsentiert, gemäß den Standardkonventionen für Namen von Variablen.
Typ	Datentyp des an die Prozedur zu übergebenden Arguments. Zulässige Typen

© by Josef Broukal, Vienna, Austria

Teil	Beschreibung
	sind Boolean, Integer, Long, Currency, Single, Double, Date, String (nur Zeichenfolgen variabler Länge), Object, Variant, ein benutzerdefinierter Typ oder ein Objekttyp.

Aufruf

Der Aufruf einer Sub-Prozedur erfolgt durch Angabe des Prozedurnamens, gefolgt von der Argumentliste. Es empfiehlt sich aber, das Befehlswort „Call“ vor den Prozedurnamen zu setzen—so weiß man immer, daß man eine Prozedur aufruft, und nicht ein VBA-Objekt.

```
Sub AufrufendeProzedur()
  Call AusgabeImTestfenster("Hallo Welt")
End Sub

Sub AusgabeImTestfenster(sText)
  Debug.Print sText
End Sub
```

Hinweis In der VBA–Hilfe finden Sie unter „Call“ genauere Informationen zum Aufrufen von Sub-Prozeduren.

Beispiel für „Argumentliste“

```
Sub HauptProzedur()

iEingabe = 20
Call TochterProzedur(iEingabe, "Guten Morgen")

End Sub

Sub TochterProzedur(iText, sGrussformel)

End Sub
```

iEingabe: Zahl, deren Adresse im Computer-Speicher an die Variable „iText“ in der Tochterprozedur weitergegeben wird. Wird „iZahl“ verändert, so ändert auch „iEingabe“ seinen Wert. („Übergabe per Referenz“.)

Guten Morgen: Da keine Variable, sondern ein Text, wird an „sGrussformel“ der Text weitergegeben. („Übergabe als Wert“.)

Übergabe als Wert erzwingen

Variable und Datenfelder werden immer per Referenz übergeben, es sei denn, Sie ordnen die Übergabe als Wert an.

```
Sub TochterProzedur(byVal iText, sGrussformel)

End Sub
```

„ByVal“ wird vor de Namen der Variablen in der auffangenden Prozedur gestellt. Damit geht der Zusammenhang zwischen „iEingabe“ und „iText“ verloren. Änderungen des Wertes von „iText“ haben auf den Wert von „iEingabe“ keine Auswirkung mehr.

© by Josef Broukal, Vienna, Austria

Anmerkungen

Sub-Prozeduren sind standardmäßig Public, wenn sie nicht explizit mit Private festgelegt werden. Wird Static nicht angegeben, so bleiben die Werte lokaler Variablen zwischen den Aufrufen nicht erhalten.

Der gesamte ausführbare Code muß sich in Prozeduren befinden. Innerhalb einer Sub-Prozedur kann keine weitere Sub-Prozedur definiert werden.

Die Anweisung „Exit Sub“ führt zum unmittelbaren Verlassen einer Sub-Prozedur. Die Programmausführung wird mit der Anweisung fortgesetzt, die auf die Anweisung mit dem Aufruf der Sub-Prozedur folgt. „Exit Sub“-Anweisungen können beliebig oft an beliebigen Stellen innerhalb einer Sub-Prozedur verwendet werden.

Im Gegensatz zu einer Function-Prozedur (die einen Wert zurückgibt) kann eine Sub-Prozedur nicht in einem Ausdruck verwendet werden.

Nicht möglich: Variable = Call EineProzedur Argument:=, …
Möglich: Variable = EineFunktion (Argument:=, …)

Variablen in Sub-Prozeduren können entweder explizit deklariert oder ohne Deklaration verwendet werden. Explizit (über Dim) in einer Prozedur deklarierte Variablen sind innerhalb dieser Prozedur immer lokal. Andere Variablen, die in einer Prozedur verwendet, aber dort nicht deklariert werden, sind auch lokal, sofern sie nicht bereits explizit auf einer höheren Ebene außerhalb der Prozedur deklariert wurden.

Vorsicht Eine Prozedur kann eine nicht explizit in der Prozedur deklarierte Variable zwar verwenden, aber es kann zu Namenskonflikten kommen, falls ein anderes auf Modulebene definiertes Element denselben Namen hat. Verwendet Ihre Prozedur eine nicht deklarierte Variable mit einem Namen, der dem Namen einer anderen Prozedur, Konstanten oder Variablen (auf Modulebene) entspricht, so wird die Variable als Referenz auf den Namen interpretiert, der auf Modulebene definiert wurde.

Hinweis Deklarieren Sie Variablen, um solche Konflikte zu vermeiden. Mit der Anweisung Option Explicit am Beginn eines Modulblattes oberhalb der ersten Prozedur können Sie die explizite Deklaration von Variablen erzwingen:

```
Option Explicit

Sub ErsteProzedur
  Dim ...
  ...
End Sub
```

Nicht bekannte Zahl von Argumenten übergeben

Angenommen, Sie können während des Programmierens nicht wissen, viewiele Argumente eine Prozedur zur Laufzeit des Programmes empfangen wird. In diesem Fall müssen Sie diese Argumente in einem Variant-Datenfeld als „Argumentliste“ übergeben. Die Prozedur muß dann selbst die Anzahl und Bedeutung der Argumente herausfinden. Ein Beispiel:

© by Josef Broukal, Vienna, Austria

```
Sub HauptProzedur()
  zeingabe = 20
  Call TochterProzedur(zeingabe, "Guten Morgen", "Guten Abend")
End Sub

Sub TochterProzedur(ParamArray AuffangendesDatenfeld())
  For zähler = LBound(AuffangendesDatenfeld) _
    To UBound(AuffangendesDatenfeld)
    MsgBox AuffangendesDatenfeld(zähler)
  Next
End Sub
```

Das Gleiche gilt für Funktionen.

Die VBA–Hilfe gibt eine vollständige Darstellung aller Aspekte des Verwendens von Prozeduren in VBA. Suchen Sie zunächst den Index–Eintrag „Prozedur“ und wählen Sie dann aus den angebotenen Themen „Prozedur-Anweisung“.

Funktionen

Eine Funktion ist eine selbständige Einheit von Anweisungen, die Argumente erhalten, eine Reihe von Anweisungen ausführen und die Werte der übergebenen Argumente ändern kann. Sie entspricht damit einer Prozedur, allerdings mit drei Unterschieden:

- eine Funktion kann niemals ein VBA–Programm starten (wohl aber von einem Tabellenblatt aus gestartet werden)
- eine Funktion liefert an die aufrufende Anweisung einen Wert zurück
- eine Funktion kann den Sprachumfang von EXCEL erweitern und wie eine eingebaute Funktion in Tabellenblättern eingesetzt werden.

Im Gegensatz zu einer Sub–Prozedur kann eine Funktions-Prozedur auf der rechten Seite eines Ausdrucks verwendet werden. Dadurch läßt sich der Rückgabewert der Funktions-Prozedur genau so weiter verarbeiten wie der Rückgabewert einer integrierten Funktion, zum Beispiel Wurzel, Cos oder Zn. Der Rückgabewert wird durch Zuweisung an den Namen der Funktion bereitgestellt.

Der Aufruf einer Funktions-Prozedur innerhalb eines Ausdrucks erfolgt durch Angabe des Funktionsnamens, gefolgt von der Argumentliste in Klammern. Die Klammern müssen auch dann angegeben werden, wenn die Funktion keine Argumente hat.

```
Resultat=Funktionsprozedur ()
Resultat=Funktionsprozedur (Arg1, ... Argn)

Function Wochennummer(Prüfdatum As Date) As Variant

Dim ErsterJänner As Long
ErsterJänner = DateSerial(Year(Prüfdatum), 1, 1)
Wochennummer = Prüfdatum - ErsterJänner + WeekDay(ErsterJänner) - 2
Wochennummer = Fix(Wochennummer / 7 + 1)

End Function
```

Diese Funktion berechnet die Wochennummer eines übergebenen Datums. Das Datum kann als Datumszahl, als Text oder als Bezug auf eine Zelle, die ein Datum enthält, eingegeben werden. Und so wird diese Funktion eingesetzt:

© by Josef Broukal, Vienna, Austria

```
LaufendeWoche = Wochennummer(“4. April 1994“)
```

Die folgende Übersicht zeigt Ihnen den grundsätzlichen Aufbau einer VBA–Funktion.

Syntax

```
[Public | Private][Static] Function Name [(ArgListe)][As Typ]

  [Anweisungen]
  [Name = Ausdruck]
  [Exit Function]
  [Anweisungen]
  [Name = Ausdruck]

End Function
```

Elemente

Die Function-Anweisung besteht aus folgenden Teilen:

Teil	Beschreibung
Public	Die Funktion kann in jedem Modulblatt eines Projektes angesprochen werden..
Private	Auf die Funktions-Prozedur kann nur durch andere Prozeduren aus dem Modul zugegriffen werden, in dem sie selbst steht.
Static	Alle innerhalb der Funktion geschaffenen (lokalen) Variablen behalten nach Abarbeiten der Funktion ihren Wert bei
Name	Name der Funktion, gemäß den Standardkonventionen für Namen von Variablen.
ArgListe	Variablenliste mit den Argumenten, die an die Funktions-Prozedur beim Aufruf übergeben werden. Mehrere Variablen werden durch Beistriche voneinander getrennt.
Typ	Datentyp des an die Prozedur zu übergebenden Arguments. Zulässige Typen sind Boolean, Integer, Long, Currency, Single, Double, Date, String (nur Zeichenfolgen variabler Länge), Object, Variant, ein benutzerdefinierter Typ oder ein Objekttyp. Datenfelder lassen sich (unabhängig von ihrem Typ) nicht zurückgeben, bei Variant-Werten, die ein Datenfeld enthalten, ist dies jedoch möglich.
Anweisungen	Eine beliebige Gruppe auszuführender Anweisungen im Rumpf einer Funktions-Prozedur.
Ausdruck	Rückgabewert der Funktion.

Die Syntax für das Argument ArgListe ist:

```
[Optional][ByVal | ByRef][ParamArray] Variablenname[( )][As Typ]
```

Die Teile von ArgListe sind:

© by Josef Broukal, Vienna, Austria

Teil	Beschreibung
Optional	Ein Argument muß nicht angegeben werden. Alle im Anschluß an Optional in ArgListe angegebenen Argumente müssen auch optional sein und mit dem Schlüsselwort Optional deklariert werden. Alle als Optional angegebenen Argumente müssen den Typ Variant haben. Optional kann nicht verwendet werden, wenn ParamArray verwendet wird.
ByVal	Das Argument wird als Wert übergeben.
ByRef	Das Argument wird als Referenz übergeben.
ParamArray	Ist nur als letztes Argument in ArgListe zulässig und gibt an, daß das letzte Element ein als Optional deklariertes Datenfeld mit Variant-Elementen ist. Das Schlüsselwort ParamArray erlaubt die Angabe einer variablen Anzahl von Argumenten und darf nicht in Kombination mit den Schlüsselwörtern ByVal, ByRef oder Optional verwendet werden. Ein Beispiel dafür wurde bereits auf Seite 27 gegeben.
Variablenname	Name der Variablen, die das Argument repräsentiert, gemäß den Standardkonventionen für Namen von Variablen.
Typ	Datentyp des an die Prozedur zu übergebenden Arguments. Zulässige Typen sind Boolean, Integer, Long, Currency, Single, Double, Date, String (nur Zeichenfolgen variabler Länge), Object, Variant, ein benutzerdefinierter Typ oder ein Objekttyp.

Rückgabe des Funktionsergebnisses

Funktionen berechnen etwas und stellen das Ergebnis der Berechnung dem Programm zur Verfügung. Innerhalb der Funktion müssen Sie das Rechenergebnis einer Variablen zuweisen, die den Namen der Funktions-Prozedur hat. Solche Zuweisungen sind beliebig oft an beliebigen Stellen innerhalb der Funktion möglich. Wenn dem Funktionsnamen kein Wert zugewiesen wurde, liefert die Funktion einen Standardwert: Eine numerische Funktion liefert 0, eine Zeichenfolgenfunktion liefert eine leere Zeichenfolge (""), und eine Funktion mit dem Rückgabetyp Variant liefert „Empty“ („Leer“). Eine Funktion, die eine Objektreferenz zurückgibt, liefert „Nothing“ („Nichts“), wenn dem Funktionsnamen (unter der Verwendung der Anweisung „Set“) innerhalb der Funktion keine Objektreferenz zugewiesen wurde.

Ein Beispiel dafür auf Seite 28.

Die VBA–Hilfe gibt eine vollständige Darstellung aller Aspekte des Verwendens von Funktionen in VBA. Suchen Sie zunächst den Index–Eintrag „Funktion“ und wählen Sie dann aus den angebotenen Themen „Funktion–Anweisung“.

Gültigkeitsbereich von Prozeduren und Funktionen

Prozeduren und Funktionen sind standardmäßig Öffentlich, wenn sie nicht explizit mit Privat festgelegt werden. Wird (zusätzlich) Statisch angegeben, so bleiben die Werte lokaler Variablen zwischen den Aufrufen der Funktion oder Prozedur erhalten.

Hinweis Damit Sie sicher auf Prozeduren und Funktionen in fremden Arbeitsmappen zugreifen können, müssen Sie in der Arbeitsmappe, in der Sie diese aufrufen, einen „Verweis“ eintragen. Sie erreichen das Dialogfeld „Verweise“ über das Menü „EXTRAS“, Befehl „Verweise...“. Dieser Befehl ist nur sichtbar, wenn ein Modulblatt aktiv ist!

© by Josef Broukal, Vienna, Austria

Anweisungen

Eine Anweisung ist die kleinste Einheit von Code und Operatoren, die eine Aktion, Definition oder Deklaration ermöglichen. Beispiele dafür sind:

`Wert1=5`	Zuweisung eines Wertes an eine Variable
`Produkt=Wert1*Wert2`	Multiplikation zweiter Werte mit Hilfe des Operators „*“; Zuweisung des Ergebnisses an eine Variable
`x=Application.Max(5,12)`	Aufruf der EXCEL–Funktion „Max“; Übergabe zweier Werte; Zuweisung des Ergebnisses an eine Variable
`Open „TESTDAT.XLS“`	Anweisung, die Datei zu öffnen
`Unterprozedur Wert1, Wert2`	Starten einer Prozedur und Übergabe von zwei Werten an diese
`y=MeineFunktion(a,b,c)`	Aufrufen einer Funktion und Übergabe von drei Werten an diese; Zuweisen des Ergebnisses der Berechnungen innerhalb der Funktion an eine Variable
`DIM MeineVariable As Integer`	Deklaration eines Variablennamens und Zuweisen eines Datentyps an die Variable

Mehrere Anweisungen pro Zeile Dies ist zwar schlechter Programmierstil, weil es Code unübesichtlich macht, es ist aber möglich, wenn Sie nach jeder Anweisung einen Doppelpunkt setzen.

```
Wert1 = 5: Wert2 = 8: Produkt = Wert1 * Wert2
```

Anweisungen, die wegen ihrer Länge mehr als eine Zeile Code brauchen Setzen Sie an das Ende jeder Zeile, in der Sie die Anweisung nicht zu Ende schreiben konnten, ein Leerzeichen gefolgt von einem Unterstrich.

```
Unterprozedur Wert1, _
    Wert2
```

Variablen und Datentypen

In den Beispielen zum Abschnitt „Anweisungen“ haben wir den Begriff „Variable“ kennengelernt. Was ist das eigentlich?

Eine Variable ist eine Speicherstelle im Computer, die Daten aufnehmen kann, die von einem VBA–Programm erfragt und auch geändert werden können. Jede Variable hat einen eindeutigen Namen, der sie innerhalb ihres Gültigkeitsbereiches identifiziert.

Syntax

```
[Dim | Private | Public] Variablenname[([Indizes])][As Typ]
```

Elemente

Die Deklarations-Anweisung einer Variablen besteht aus folgenden Teilen:

© by Josef Broukal, Vienna, Austria

Teil	Beschreibung
Variablenname	Name der Variablen, gemäß den Standardkonventionen für Namen von Variablen.
Indizes	Bei Datenfelder die Anzahl der Element-Dimensionen und der Elemente pro Dimension.
Typ	Datentyp der Variablen. Boolean, Integer, Long, Currency, Single, Double, Date, String, String * Anzahl (für Zeichenfolgen fester Länge), Object, Variant, ein anderer benutzerdefinierter Typ oder ein Objekttyp.

Die Namen von Variablen müssen mit einem Buchstaben beginnen; sie dürfen keine Leerzeichen enthalten:

Falsch: `5Zeilen  Meine Variable`
Richtig: `Fünf_Zeilen  MeineVariable`

Die folgende Tabelle enthält die von Visual Basic unterstützten Datentypen sowie deren Speicherbedarf und Wertebereiche.

Datentyp	Speicherbedarf	Bereich
Boolean	2 Bytes	True oder False (Zahl <>0 oder 0)
Integer	2 Bytes	-32.768 bis 32.767.
Long(lange Ganzzahl)	4 Bytes	-2.147.483.648 bis 2.147.483.647.
Single (Gleitkommazahl mit einfacher Genauigkeit)	4 Bytes	-3,402823E38 bis -1,401298E-45 für negative Werte. 1,401298E-45 bis 3,402823E38 für positive Werte.
Double (Gleitkommazahl mit doppelter Genauigkeit)	8 Bytes	-1,79769313486232E308 bis -4,94065645841247E-324 für negative Werte. 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte.
Currency (skalierte Ganzzahl)	8 Bytes	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807.
Date (für Datum und Zeit)	8 Bytes	01. Januar 100 bis 31. Dezember 9999.
Object	4 Bytes	Beliebiger Verweis auf ein Objekt.
String (Zeichenfolge)	1 Byte pro Zeichen	0 bis ca. 2 Milliarden (ca. 65.535 in Microsoft Windows, Version 3.1 und niedriger)
String*_Anzahl_		Maximal _Anzahl_ von Zeichen
Variant	16 Bytes + 1 Byte für jedes Zeichen	Numerische Werte im Bereich des Datentyps Doppelt oder ein beliebiger Text.
Benutzerdefiniert (mit Typ)	Anzahl ist von den Elementen abhängig	Der Bereich für jedes Element entspricht dem Bereich des zugehörigen Datentyps.

Anmerkungen

Wenn Sie keinen Daten- oder Objekttyp angeben, erhält die Variable standardmäßig den Datentyp Variant. Dieser Variablentyp kann Werte aller Datentypen und Verweise auf alle Objekttypen enthalten. Er braucht den meisten Speicherplatz und wird langsamer abgearbeitet als eine Variable, die einen konkreten Datentyp (String, Integer, Double, Date etc.) oder Objekttyp (Workbook, Range, Menubar etc.) hat.

Bei der Initialisierung wird numerischen Variablen der Wert 0 und Zeichenfolgen variabler Länge eine leere Zeichenfolge zugewiesen. Zeichenfolgen fester Länge werden mit Nullen aufgefüllt und Variant-Variablen werden mit Leer initialisiert. Alle Elemente einer Variablen mit einem benutzerdefinierten Typ werden genauso initialisiert wie Variablen entsprechenden Datentyps. Einer Variablen mit einer Referenz auf ein Objekt muß mit der Setze-Anweisung ein existierendes Objekt zugewiesen werden,

© by Josef Broukal, Vienna, Austria

bevor sie verwendet werden kann. Eine deklarierte Objektvariable hat vor dem Zuweisen eines Objekts den speziellen Wert Nothing, der anzeigt, daß die Variable auf kein Objekt verweist.

Deklaration und Gültigkeitsbereich von Variablen

Lokal—Nur in einer Prozedur oder Funktion sichtbar Wenn Sie eine Variable in einer Prozedur erzeugen:

```
Variable=Wertzuweisung
```

oder gleich unterhalb des Prozedur- oder Funktionskopfes ausdrücklich einrichten:

```
Sub EineProzedur()
  Dim Variable
End Sub
```

dann ist diese Variable „lokal“, das heißt, Sie ist nur in dieser Prozedur „sichtbar“. Verläßt Ihr Programm die Prozedur, wird die Variable im Computerspeicher gelöscht. Sie können aber festlegen, daß Ihre Variable in mehr als einer Prozedur sichtbar und ansprechbar ist.

Hinweis Wenn Sie die Dim-Anweisung innerhalb einer Prozedur verwenden, sollten Sie sie an den Anfang der Prozedur stellen. Somit bleibt sie gut sichtbar.

Privat—In allen Prozeduren und Funktionen eines Modulblattes sichtbar wird eine Variable dann, wenn Sie sie am Beginn des Modulblattes—noch vor dem Beginn der ersten Prozedur—mit einer „Private“–Anweisung deklarieren:

```
Private Variable as Integer

Sub Hauptprozedur()
    Variable=Wertzuweisung
Ende Sub
```

Wird der Modul verlassen, wird die Variable im Computerspeicher gelöscht.

Wird in einer Prozedur oder Funktion eine Variable deklariert, die den selben Namen hat wie eine private Variable im selben Modulblatt, dann „verbirgt“ diese Variable den Blick auf die modulweit gültige.

```
Private TagesDatum as Integer

Sub Hauptprozedur()
    Dim TagesDatum as Date
    Variable=Wertzuweisung
Ende Sub
```

Die in „Hauptprozedur“ geschaffene Variable „TagesDatum“ hat mit der modulweit geschaffenen nichts zu tun. Die modulweit geschaffene Variable ist in „Hauptprozedur“ aber nicht mehr sichtbar.

Öffentlich—In allen Prozeduren und Funktionen eines Projekts sichtbar, solange das zugehörige VBA-Programm läuft ist eine Variable dann, wenn Sie sie am Beginn eines Modulblattes—noch vor dem Beginn der ersten Prozedur—mit einer „Public“–Anweisung deklarieren:

© by Josef Broukal, Vienna, Austria

```
Public Variable As Integer

Sub Hauptprozedur()
    Variable=Wertzuweisung
Ende Sub
```

Wird auf Modulebene eine gleichnamige Variable definiert, dann „verbirgt“ diese Variable im ganzen Modulblatt den Blick auf die projektweit gültige. Wird in einer Prozedur oder Funktion eine gleichnamige Variable deklariert, dann „verbirgt“ diese Variable den Blick auf eine gleichnamige modulweit oder projektweit gültige.

Ein „Projekt“ umfaßt sämtlichen Code in sämtlichen Modulblättern, die für ein VBA–Programm angelegt worden sind.

Statische Variable

Mit der Anweisung „Statisch“ in einer Funktion oder Prozedur deklarierte Variable behalten ihren Wert, solange das VBA–Programm läuft.

```
Static Username as String
```

Datenfelder

Datenfelder sind eine besondere Art von Variablen. Sie sind eine Menge aufeinanderfolgender indizierter (das heißt mit einer fortlaufender Nummer ansprechbarer) Elemente mit einem gemeinsamen Datentyp. Wenn Sie wollen, daß die einzelnen Elemente eines Datenfeldes verschiedene Datentypen aufnehmen können, dann deklarieren Sie das Datenfeld als Typ „Variant“. Jedes Datenfeldelement wird durch eine eindeutige Indexnummer gekennzeichnet. Änderungen an einem Element bleiben ohne Wirkung auf die anderen Elemente. Datenfelder werden mit der „Dim“–, „Private“ – oder „Public“–Anweisung deklariert.

Syntax

```
Dim Variablenname[([Indizes])][As Typ]
```

Indizes bestimmen die Dimensionen einer Datenfeldvariablen. Bis zu 60 Dimensionen gleichzeitig sind zulässig. Die Syntax für das Argument Indizes ist:

```
[Untergrenze To] Obergrenze [,[Untergrenze To] Obergrenze]
```

Die Anweisung „Option Base“

Das erste Element eines Datenfeld erhält standardmäßig den Index 0. Das ist oft unpraktisch—zum Beispiel, weil es in Tabellen weder Spalte noch Zeile 0 gibt. Mit der Anweisung

```
Option Base 1
```

am Beginn eines Modulblattes erreichen Sie, daß alle Datenfelder, die in diesem Modul geschaffen werden, als ersten Index 1 haben.

© by Josef Broukal, Vienna, Austria

Beispiele

STARRE ANZAHL VON ELEMENTEN

Wenn Sie ein Datenfeld deklarieren und gleichzeitig die Anzahl seiner Elemente festlegen, dann kann die Größe des Datenfeldes während des Programmlaufes nicht verändert werden.

TagFeld ist ein Datenfeld mit Variant-Werten und 51 indizierten Elementen von 0 bis 50, vorausgesetzt, Option Base wurde für das aktuelle Modul auf 0 (Voreinstellung) gesetzt.

```
Dim TagFeld(50)
```

Matrix ist ein zweidimensionales Datenfeld mit ganzen Zahlen.

```
Dim Matrix(3,4) As Integer
```

TestMatrix ist ein dreidimensionales Datenfeld mit Werten vom Typ Double und explizit angegebenen Grenzen.

```
Dim TestMatrix(1 To 5,  4 To 9,  3 To 5) As Double
```

Geburtstag ist ein Datenfeld mit Datumsangaben und Indizes von 1 bis 10.

```
Dim Geburtstag(1 To 10) As Date
```

DYNAMISCHES FESTLEGEN DER ELEMENTEZAHL

Die Dim– , Private– oder Public–Anweisung in Kombination mit einem leeren Klammernpaar dient der Deklaration dynamischer Datenfelder. Nach einer solchen Deklaration können Sie mit der Redim-Anweisung innerhalb einer Prozedur die Anzahl der Dimensionen und Elemente definieren. Wenn Sie eine Dimension für eine Datenfeldvariable mit einer weiteren Deklaration ändern wollen und die Größe des Datenfelds bereits explizit mit einer Dim– , Private– oder Public–Anweisung festgelegt wurde, tritt ein Fehler auf.

Beispiele

TestFeld ist ein dynamisches Datenfeld. Es wird—ohne Festlegung der Zahl der Elemente—so deklariert:

```
Dim TestFeld() As Integer
```

Später wird im Code dann (innerhalb einer Prozedur oder Funktion) die Anzahl der Elemente festgelegt:

```
Redim TestFeld(5)  ' 5 Elemente reservieren.
For I = 1 To 5  ' 5 Schleifendurchläufe.
  TestFeld(I) = I  ' Datenfeld initialisieren.
Next I
```

Die folgende Anweisung ändert die Größe des Datenfelds und löscht die übernommenen (d.h. die ersten fünf) Elemente.

© by Josef Broukal, Vienna, Austria

```
Redim TestFeld(10)  ' Größe auf 10 Elemente ändern.
For I = 1 To 10  ' 10 Schleifendurchläufe.
  TestFeld(I) = I  ' Datenfeld initialisieren.
Next I

' Die nächste Anweisung ändert die Größe des Datenfelds, löscht die Elemente aber nicht.

Redim Preserve TestFeld(15)  ' Größe auf 15 Elemente ändern.
```

Gültigkeitsbereich von Datenfeldern

Datenfelder, die mit „Public“ auf Modulebene deklariert wurden, stehen allen Prozeduren und Funktionen in allen Modulen zur Verfügung. Datenfelder, die mit „Private“ auf Modulebene deklariert wurden, stehen allen Prozeduren des Moduls zur Verfügung. Auf Prozedur- oder Funktionsebene deklarierte Datenfelder stehen nur innerhalb der sie umgebenden Prozedur oder Funktion zur Verfügung.

Wenn Sie keinen Datentyp angeben, erhält das Datenfeld standardmäßig den Datentyp Variant.

Bei der Initialisierung wird numerischen Datenfeldelementen der Wert 0 und Zeichenfolgen variabler Länge eine leere Zeichenfolge zugewiesen. Zeichenfolgen fester Länge werden mit Nullen aufgefüllt und Variant-Variablen werden mit „Empty“ initialisiert.

Die „Array“–Funktion

Liefert einen Variant mit einem Datenfeld.

Syntax

```
Variant=Array(ArgListe)
```

ArgListe enthält eine durch Beistriche getrennte Liste mit einer unbestimmten Anzahl von Werten, die dem im Variant enthaltenen Datenfeld als Elemente zugewiesen werden. Werden keine Argumente angegeben, so erstellt die Funktion ein Datenfeld mit der Länge Null.

```
Dim A As Variant
A = Array(10,20,30)
B = A(2)
```

Hinweis Die untere Grenze für ein mit der Datenfeld-Funktion erstelltes Datenfeld ist durch die „Option Base“-Anweisung festgelegt. Gilt auf dem Modulblatt „Option Base 1“ so enthält A(2) den Wert 20, bei „Option Base 0“ den Wert 30.

Anmerkungen

Obwohl sich ein Variant mit einem darin enthaltenen Datenfeld konzeptionell von einem Datenfeld mit Elementen des Typs Variant unterscheidet, wird auf die Datenfeldelemente in derselben Weise zugegriffen. Der Zugriff auf ein Element eines Datenfelds erfolgt durch Angabe des Variablennamens mit einem nachfolgenden Klammernpaar, das die Indexzahl des gewünschten Elements enthält. Im vorangehenden Beispiel erstellt die erste Anweisung die Variable A als einen Variant. Die zweite Anweisung weist der Variablen A ein Datenfeld zu. Die letzte Anweisung verdeutlicht, wie das zweite Datenfeldelement einer anderen Variablen zugewiesen werden kann.

© by Josef Broukal, Vienna, Austria

Datenfeld-Variant mit Inhalt eines Zellbereiches füllen

Liegt in „Tabelle1“ ein zweitspaltiger Zellbereich vor, der in jeder Spalte eine Liste enthält:

	A	B
1	Anton	Januar
2	Ärger	Februar
3	Berta	März
4	Cäsar	April
5	Dora	Mai
6	Emil	Juni
7	Friedrich	Juli
8	Gustav	August
9	Heinrich	September
10	Ida	Oktober
11	Josef	November
12	Konrad	Dezember
13	Ludwig	Januar

und ordnen wir mittels VBA an:

```
Sub try()

Dim vNamensEinträge As Variant

vNamensEinträge = ThisWorkbook.Sheets("Tabelle1"). _
  Cells(1, 1).CurrentRegion

End Sub
```

so enthält der Variant „vNamensEinträge“ ein zweidimensionales Datenfeld mit den Inhalten der Spalten des Zellbereiches:

Datenfeld-Variant
1. Element der ersten Ordnung (leer)
2. Element der ersten Ordnung (leer)
Letztes Element der ersten Ordnung (leer)
1. Element der zweiten Ordnung (Anton)
2. Element der zweiten Ordnung (Januar)
1. Element der zweiten Ordnung (Ärger)
2. Element der zweiten Ordnung (Februar)
1. Element der zweiten Ordnung (Zeppelin)
2. Element der zweiten Ordnung (April)

Wert von Spalte „A“ in Zeile „1“ erfragen Dazu schreiben wir:

```
MsgBox vNamensEinträge(1,1)
```

Und bei der Frage nach dem Wert von Spalte „B“ in Zeile „1“:

```
MsgBox vNamensEinträge(1,2)
```

Anzahl der Elemente erfragen Die Zahl der Elemente 1. Ordnung (=Anzahl der Datenzeilen in der Tabelle) erfragen wir so:

© by Josef Broukal, Vienna, Austria

```
MsgBox "Anzahl der Datenzeilen= " & UBound(vNamensEinträge)
```

Und die Anzahl der Elemente 2. Ordnung pro Element erster Ordnung so:

```
MsgBox "Anzahl der Datenspalten= " & UBound(vNamensEinträge, 2)
```

Hinweis Das niedrigste Element eines mit einem Zellbereich gefüllten Datenfeld-Variants ist immer Eins, auch wenn „Option Base 0“ angeordnet wurde.

Den Code dieses Beispiels finden Sie im Verzeichnis „Beispiel“ in der Datei „VARIANT.XLS“.

Konstanten

Während Variable und Datenfelder ihre Werte ändern können, während ein VBA-Programm läuft, behalten Konstante ihren Wert unabänderlich. Jeder Versuch, ihnen einen neuen Wert zu geben, führt zu einer Fehlermeldung und zum Abbruch des Programmes.

Deklaration von Konstanten

Innerhalb einer Prozedur werden Konstante so eingerichtet:

```
Sub MeineProzedur()
  Const DATUMS_SPALTE=1
  ...
End Sub
```

Auf Modulebene richten Sie Konstante ein, indem Sie sie noch vor dem Beginn der ersten Prozedur oder Funtion mit dem Befehlswort „Private“ deklarieren. Diese Konstanten sind dann in allen Prozeduren und Funktionen auf dem Modulblatt bekannt. Richten Sie in einer Prozedur oder Funktion eine gleichnamige Konstante ein, dann „vebirgt“ diese die modulweit deklarierte:

```
Private Const DATUMS_SPALTE=1

Sub MeineProzedur()
  Cells(7, DATUMS_SPALTE).Select
End Sub
```

Global sichtbar ist eine Konstante dann, wenn Sie sie auf irgendeinem Modulblatt eines Projektes noch vor dem Beginn der ersten Prozedur oder Funtion mit dem Befehlswort „Public“ deklarieren. Diese Konstanten sind dann in allen Prozeduren und Funktionen auf allen Modulblättern bekannt. Richten Sie auf einem Modulblatt eine private gleichnamige Konstante ein, so „verbirgt“ sie gegenüber allen Funktionen und Prozeduren auf diesem Blatt die projektweit deklarierte. Richten Sie in einer Prozedur oder Funktion eine gleichnamige Konstante ein, dann überdeckt diese innerhalb der Prozedur oder Funktion die globale Konstante.

```
Public Const HOECHSTGRENZE=1000

Sub MeineProzedur()
  If iEinWert > HOECHSTGRENZE Then …
End Sub
```

Hinweis Es hat sich eingebürgert, den Namen von Konstanten ausschließlich mit Großbuchstaben zu schreiben. Sie fallen damit in einem Codetext deutlich auf und es ist jederzeit klar, daß es sich nicht um eine Variable handelt.

© by Josef Broukal, Vienna, Austria

Variablen werden üblicherweise mit Groß- und Kleinbuchstaben geschrieben.

Operatoren

Operatoren leisten die Arbeit, die VBA mit Werten oder Objekten durchführt. Wir unterscheiden:

Arithmetische Operatoren

*	Multiplikation
/	Division
+	Addition
-	Subtraktion
\	Division mit ganzzahligem Ergebnis; Nachkommastellen werden abgeschnitten
^	Potentieren
Mod	Liefert den Rest des gerundeten Ergebnisses einer Division

Logische Operatoren

Eqv	Übereinstimmung zweier Ausdrücke
Imp	Logische Implikation zweier Ausdrücke
Not	Logische Verneinung eines Ausdruckes
Or	Entweder–Oder oder beide zweier Ausdrücke
And	Logische Gleichheit zweier Ausdrücke
Xor	Logische Ausschließung zweier Ausdrücke

Vergleichsoperatoren

Syntax

```
Ergebnis = Ausdruck1 VergleichsOperator Ausdruck2
Ergebnis = Objekt1 Is Objekt2
Ergebnis = Zeichenfolge Like Muster
```

Elemente

Vergleichsoperatoren bestehen aus den folgenden Teilen:

Teil	Beschreibung
Ergebnis	Beliebige numerische Variable
Ausdruck	Beliebiger Ausdruck
VergleichsOperator	Beliebiger Vergleichsoperator
Objekt1	Beliebiger Objektname
Objekt2	Beliebiger Objektname
Zeichenfolge	Beliebiger Zeichenfolgenausdruck
Muster	Beliebiger Zeichenfolgenausdruck oder Bereich von Zeichen

Die Operatoren Is und Like haben eine spezielle Bedeutung. Is untersucht, ob zwei Objekt-Variable auf das selbe Objekt zeigen (funktioniert in EXCEL 5.0 nicht).

© by Josef Broukal, Vienna, Austria

```
Sub try()

Set Testobjekt = ThisWorkbook.Sheets(1)
Set Testobjekt3 = Application.Toolbars

Set TestObjekt2 = Testobjekt  ' Objektverweise zuweisen.
Set TestObjekt4 = Testobjekt
Set TestObjekt5 = Testobjekt3
EinTest = TestObjekt2 Is TestObjekt4  ' Liefert Wahr.
EinTest = TestObjekt5 Is TestObjekt4  ' Liefert Falsch.
EinTest = Testobjekt Is TestObjekt5  ' Liefert Falsch.

End Sub
```

Like gibt an, ob eine Zeichenfolge in einer anderen enthalten ist.

```
EinTest = "aBBBa" Like "a*a"  ' Liefert Wahr.
EinTest = "F" Like "[A-Z]"  ' Liefert Wahr.
EinTest = "F" Like "[!A-Z]"  ' Liefert Falsch.
EinTest = "a2a" Like "a#a"  ' Liefert Wahr.
EinTest = "aM5b" Like "a[L-P]#[!c-e]"  ' Liefert Wahr.
EinTest = "BAT123khg" Like "B?T*"  ' Liefert Wahr.
EinTest = "CAT123khg" Like "B?T*"  ' Liefert Falsch.
```

Hinweis Beim Vergleich von deutschen Zeichenfolgen, die Umlaute enthalten, sowie von Zeichenfolgen, in denen einmal Großschreibung und das andere Mal Kleinschreibung vorliegt, liefert „Like“ nur dann das richtige Ergebnis, wenn am Beginn des Modulblattes noch vor der ersten Prozedur oder Funktion „Option Compare Text“ angeordnet wird.

Verkettungsoperatoren

+	Hängt zwei Zeichenketten aneinander; addiert zwei Zahlen
&	Hängt zwei Zeichenketten aneinander; wandelt Zahl in Zeichenkette um

Objekte

VBA ist „objektorientiert“, aber was heißt das konkret? Alles, was wir in EXCEL sehen, bestimmen, ändern, hinzufügen oder löschen können, sind „Objekte“, die *Eigenschaften* haben und mit *Methoden* manipuliert werden können.

Mehr über Objekte im nächsten Kapitel des Skriptums.

Gültigkeitsbereich von Objekten

Objekte sind global sichtbar. Mehr über sie in den folgenden beiden Kapiteln.

Schlüsselwörter nach Aufgabenbereichen

In der VBA–Hilfe wird eine ausführliche Übersicht über alle Schlüsselwörter dieser Programmiersprache gegeben. Viele Hyptertext-Verknüpfungen machen Ihnen die Hilfeseiten zu den einzelnen Schlüsselwörtern zugänglich.

© by Josef Broukal, Vienna, Austria

Die Schlüsselwörter–Übersicht finden Sie in der EXCEL-Online-Hilfe zu EXCEL 5.0 unter „Inhalt der Microsoft EXCEL–Hilfe“—„Programmieren mit Visual Basic“—„Schlüsselwörter nach Aufgabenbereichen“, in der Hilfe zu EXCEL 7.0 im Register „Suchen“ des Hilfe-Dialogfeldes nach Eingabe des Suchbegriffes „Schlüsselwörter“.

© by Josef Broukal, Vienna, Austria

Objekte 1: Übersicht

Objekt sind die Grundbausteine von Microsoft Excel. Angesprochen, verändert, ausgelesen werden sie mit dem Visual-Basic-Code. VBA-Programme verwenden Objekte, um Aufgaben in Microsoft Excel durchzuführen, in ähnlicher Weise, wie Sie Objekte im Alltag verwenden. Beispielsweise ist ein LKW ein Objekt, das Güter transportiert, und eine Tabelle ist ein Objekt, das Informationen berechnet und anzeigt.

In der nachstehenden Tabelle sind einige für EXCEL typische Objekte aufgeführt.

Objekt	Beschreibung
Workbook	Eine Gruppe von Blättern, die zusammen als Datei gespeichert werden.
Range	Eine Zelle oder ein Zellbereich in einer Tabelle.
Style	Eine Formatkombination für eine Zelle, mit der Schriftart, Farbe, Rahmen usw. der Zelle festgelegt werden.
Menu	Eine Liste von Befehlen, mit denen Sie Aktionen durchführen.
Name	Ein leicht zu merkender Kennzeichner, der sich auf eine Zelle, einen Zellbereich, einen Wert oder eine Formel bezieht.
Worksheet	Ein Blatt in einer Arbeitsmappe, in dem Informationen berechnet und angezeigt werden.

Wie Objekte im Alltag bestehen auch viele Objekte in Microsoft Excel aus Komponenten (die selbst Objekte sind).

Übersicht

In den drei folgenden Grafiken finden Sie eine Übersicht über die wichtigsten Objekte von EXCEL

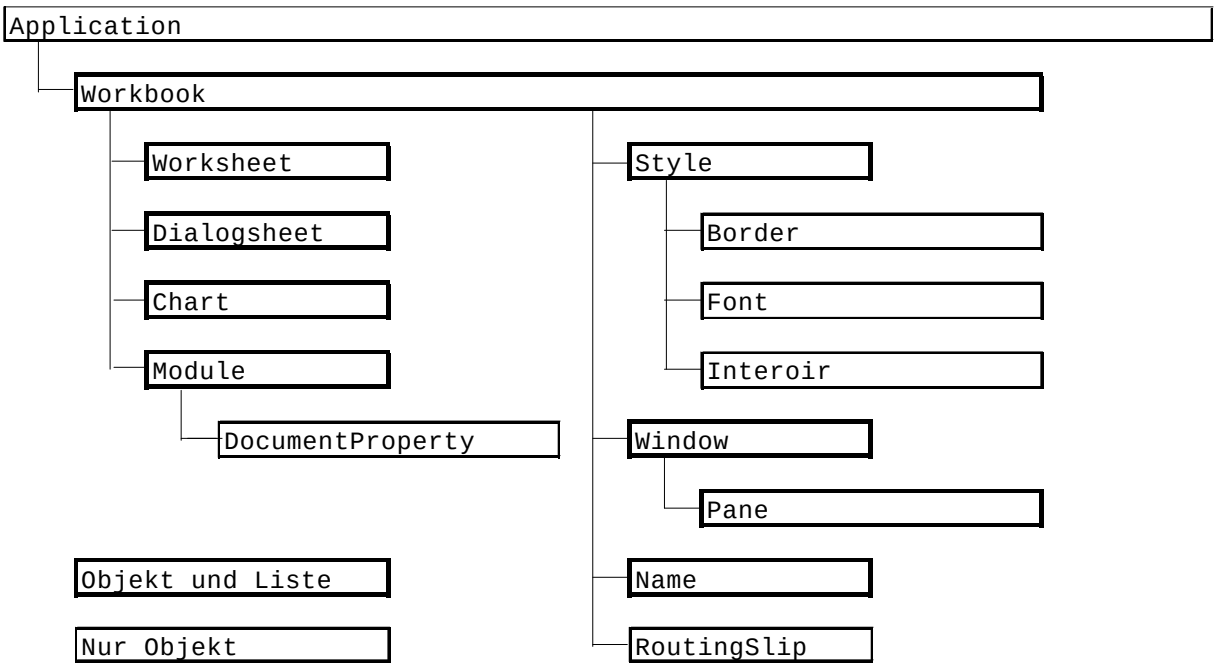


Abbildung 15: Die wichtigsten Objekte unterhalb des Application-Objektes

© by Josef Broukal, Vienna, Austria

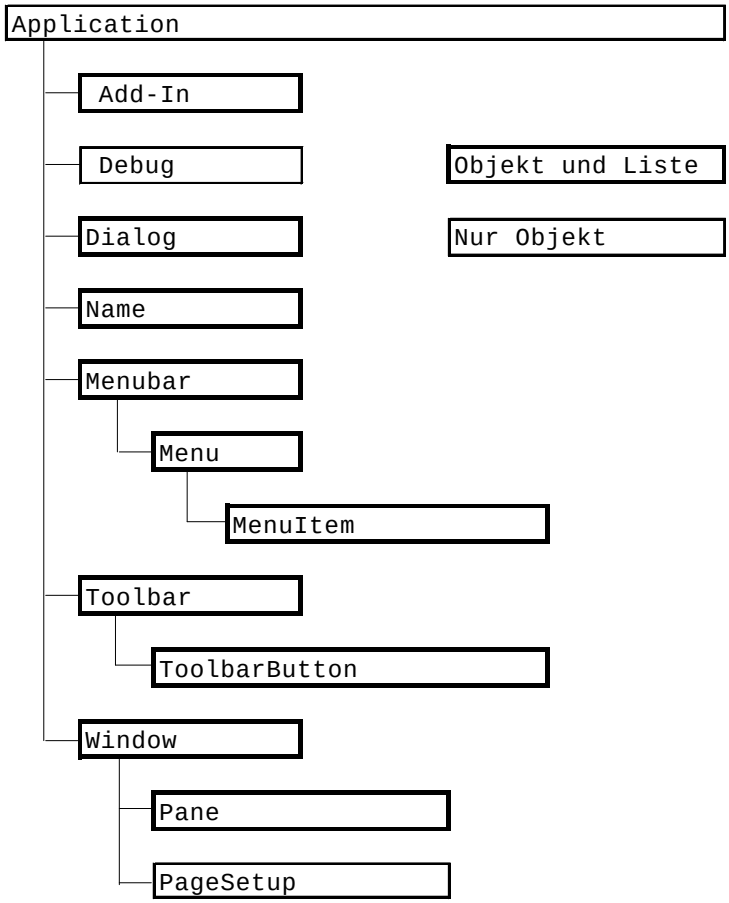


Abbildung 16: Die wichtigsten objekte unterhalb des Application-Objektes (Fortsetzung)

© by Josef Broukal, Vienna, Austria

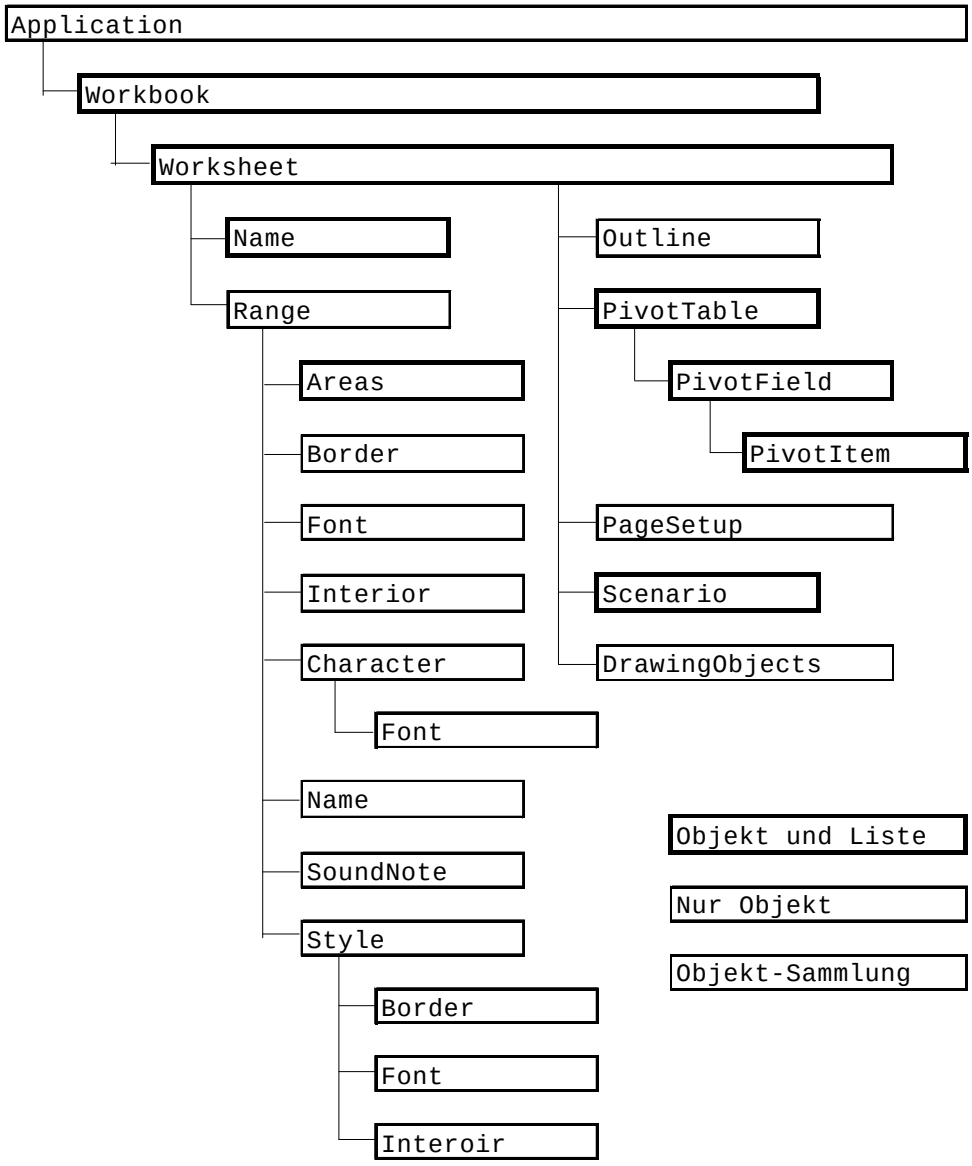


Abbildung 17: Die wichtigsten Objekte unterhalb des Worksheet-Objektes

Eigenschaften von Objekten

Eine Eigenschaft ist ein benanntes Attribut eines Objektes, das die Objektcharakteristik (zum Beispiel die Größe, Farbe oder die Lage auf dem Bildschirm) oder das Objektverhalten (zum Beispiel, ob es aktiviert oder sichtbar ist) definiert. Zum Ändern eines Objektes ändern Sie die Werte seiner Eigenschaften.

FESTLEGEN EINES EIGENSCHAFTWERTES

Sie legen den Wert einer Eigenschaft fest, indem Sie hinter den Objektnamen einen Punkt, dann den Namen der Eigenschaft, ein Gleichheitszeichen und den neuen Eigenschaftswert angeben. Beispielsweise können Sie die Größe des Fensters ändern, in dem eine Arbeitsmappe gezeigt wird, indem Sie die Werte der Width- und der Height-Eigenschaft neu festlegen:

© by Josef Broukal, Vienna, Austria

```
Sub FensterGroesseAendern()
  If ActiveWindow.WindowState <> xlNormal Then
    ActiveWindow.WindowState = xlNormal
  End If
  NeuesMass = Application.InputBox("Welche Länge und Breite soll das
Fenster haben?")
  ActiveWindow.Width = NeuesMass
  ActiveWindow.Height = NeuesMass
End Sub
```

Einige Eigenschaften können nicht geändert werden. Das Hilfethema zu einer Eigenschaft gibt an, ob die Eigenschaft geändert (Schreib-Lese-Zugriff) oder nicht geändert (Nur-Lese-Zugriff) werden kann.

ZURÜCKGEBEN EINES EIGENSCHAFTWERTES

Sie können Informationen über ein Objekt gewinnen, indem Sie den Wert einer Objekteigenschaft erfragen. Wir haben im letzten Beispiel nach dem Wert der Eigenschaft „ActiveWindow.WindowState“ gefragt. Mit den folgenden Anweisungen wird der Wert der Notiztext-Methode an eine Variable übergeben und dann der Inhalt der Variablen in einem Dialogfeld angezeigt. Notiztext ist eine Methode des Bereich-Objektes.

```
Sub StehtEineNotizInDerZelle()
  Testmeldung = Range("A1").NoteText
  MsgBox Testmeldung
End Sub
```

Durchführen von Aktionen mit Objektmethoden

Eine Methode ist ein Befehl (Aktion), den Sie an ein Objekt senden, und den das Objekt ausführen kann. In der folgenden Anweisung wird die Select-Methode des Range-Objektes verwendet, um Zelle A1 auszuwählen. (Eine oder mehrere Zellen auswählen ist eine Aktion, die das Bereich-Objekt ausführen kann.)

```
Range("A1").Select
```

OBJEKTE ALS CONTAINER ANDERER OBJEKT

Microsoft Excel besteht aus zahlreichen Objekten (Arbeitsmappen, Tabellenblätter, Diagrammblätter usw.), von denen einige selbst wiederum Objekte enthalten. Ein Objekt kann ein anderes Objekt einfach deswegen enthalten, weil es hierarchisch "höher" als das andere Objekt ist. Eine Arbeitsmappe enthält Tabellenblätter, Tabellenblätter enthalten Zeilen und Spalten, Zeilen und Spalten enthalten verschiedene Zellbereiche.

Das Anwendungs-Objekt

Das Objekt der höchsten Ebene ist das Anwendung-Objekt („Application“). Es enthält alle anderen Objekte, die zur Anwendung gehören. Wenn Sie Eigenschaften festlegen oder Methoden des Anwendung-Objektes ausführen, beeinflußt das Ergebnis Excel insgesamt. Beispielsweise wird in der folgenden Anweisung die Beenden-Methode verwendet, um Microsoft Excel zu beenden.

© by Josef Broukal, Vienna, Austria

```
Application.Quit
```

Navigieren zwischen Objekten in einem Container

Genau so wie Sie einen Punkt verwenden, um eine Eigenschaft oder Methode eines Objektes zu identifizieren, verwenden Sie ihn auch, um sich zwischen den Objekten in einem Container zu bewegen. Beispielsweise bezeichnet die folgende Anweisung das ArbeitsmappeListe-Objekt, das sich im Anwendung-Objekt befindet, und schließt alle geöffneten Arbeitsmappen.

```
Application.Workbooks.Close
```

Auflistungen von Objekten

Auflistungen von Objekten sind ein weiteres Beispiel für Container, die andere Objekte enthalten. Eine Auflistung ist ein spezielles Objekt, das mehrere Objekte desselben Typs enthält. Im vorherigen Beispiel ist „Workbooks“ eine Auflistung, da es eine Reihe von Arbeitsmappe-Objekten enthält. Auflistungen bedeuten immer "mehr als Eins" (Worksheets, Axes, usw.).

Sie können Auflistungen daran erkennen, daß dem Namen ein Mehrzahl-„S“ angehängt wurde. Visual Basic bietet Anweisungen, die das Arbeiten mit Auflistungen vereinfachen. Beispielsweise nimmt die For Each…Next-Anweisung in einer Programmschleife alle Objekte in einer Auflistung der Reihe nach her. Mit den folgenden Anweisungen wird bei Ausführung der Schleife der Name jeder Arbeitsmappe in einem eigenen Dialogfeld angezeigt.

```
Sub ZeigeAlleNamen()
  For Each EineMappe In Application.Workbooks
    MsgBox EineMappe.Name
  Next
End Sub
```

Arbeiten mit bestimmten Objekten in Containern oder Auflistungen

Oft möchten Sie mit einem bestimmten Objekt in einer Auflistung und nicht mit der gesamten Auflistung arbeiten. Das Arbeiten mit einer Auflistung ähnelt dem Arbeiten mit einem Datenfeld. Sie bezeichnen ein Element einer Auflistung, indem Sie seinen Namen oder seine fortlaufende Nummer in Klammern angeben. In der folgenden Anweisung bezeichnet Application.Workbooks(1) die erste geöffnete Arbeitsmappe, und die Close-Methode schließt diese Arbeitsmappe.

```
Application.workbooks(1).Close
```

In der folgenden Anweisung wird ein Name (als Zeichenfolge) verwendet, um ein bestimmtes Objekt zu bezeichnen.

```
Application.workbooks("Mappe1"). Close
```

MEHR ÜBER CONTAINER UND AUFLISTUNGEN

Sie können sich in einem Arbeitsgang durch mehrere Container-Objekte hindurchbewegen. So enthält die Anwendung—EXCEL—mehrere Arbeitsmappen; jede Arbeitsmappe enthält wieder mehrere Tabellenblätter. Mit der folgenden Anweisung wird das erste Tabellenblatt in der Arbeitsmappe "Mappe6" gelöscht.

```
Application.Workbooks("Mappe6").Worksheets(1).Delete
```

Jedes Tabellenblatt enthält Zeilen. Mit der folgenden Anweisung wird die zweite Zeile in "Tabelle2" der Arbeitsmappe "Mappe6" gelöscht.

© by Josef Broukal, Vienna, Austria

```
Application.Workbooks("Mappe6"). Worksheets("Tabelle2"). _
  Rows(2).Delete
```

Jede ZeileListe einer Tabelle enthält Zellen. Mit der folgenden Anweisung wird in die erste Zelle der ersten Zeile Text eingegeben.

```
Application.Workbooks("Mappe6").Worksheets("Tabelle2"). _
  Rows(1).Cells(1).Value = "Bilanz 1995"
```

ERSTELLEN VON KÜRZEREM CODE

Das obige Beispiel kann Gott sei Dank auch kürzer formuliert werden. Wenn Sie z.B. Code in Microsoft Excel schreiben, können Sie das Schlüsselwort „Application“ immer weglassen. Visual Basic bietet darüber hinaus Eigenschaften und Methoden zum Zurückgeben von Objekten, die Sie verwenden können, um das gewünschte Objekt schnell zu identifizieren. Weitere Informationen finden Sie im nächsten Abschnitt.

Eigenschaften, die Objekte zurückgeben

Eigenschaftswerte sind in der Regel Zahlen, Text, True oder False usw., manchmal aber auch Objekte. Wenn der Rückgabewert einer Eigenschaft selbst ein Objekt ist, behandeln Sie die Eigenschaft genau so wie jedes andere Objekt. Beispielsweise ist „ActiveWorkbook“ eine schreibgeschützte Eigenschaft des „Application“-Objektes und identifiziert die aktive Arbeitsmappe, die selbst ein Objekt ist. In der folgenden Anweisung wird die „Close“-Methode verwendet, um genau die Arbeitsmappe zu schließen, auf die die „AktiveArbeitsmappe“-Eigenschaft zeigt.

```
ActiveWorkbook.Close
```

Eine Reihe von Eigenschaften gibt Objekte zurück, dazu zählen „ActiveCell“, „ActiveWindow“, „ActiveSheet“ und „Selection“. Dieses Leistungsmerkmal verkürzt Ihren Code erheblich und trägt zur besseren Lesbarkeit bei. Vergleichen Sie die Länge der ersten folgenden Anweisung mit der der zweiten. Beide bewirken das gleiche, wenn der Cursor in der anzusprechenden Zelle steht:

```
ActiveCell.Value = "Bilanz 1995"
```

```
Application.Workbooks("Mappe6").Worksheets("Tabelle2"). _
  Rows(1).Cells(1).Value = "Bilanz 1995"
```

Methoden, die Objekte zurückgeben

Methoden sind in der Regel Aktionen, die ein Objekt durchführen kann. Einige Objekte verfügen jedoch über spezielle Methoden, die ein anderes Objekt zurückgeben (gewöhnlich ein Auflistungsobjekt). Wenn eine Methode ein Objekt zurückgibt, behandeln Sie die Methode wie ein Objekt. In den vorhergehenden Beispielen zu Auflistungen sind die Schlüsselwörter „Workbooks“, „Worksheets“, „Rows“ und „Cells“ nichts anderes als Methoden, die Objekte zurückgeben.

```
Application.Workbooks("Mappe6").Worksheets("Tabelle2"). _
  Rows(1).Cells(1).Value = "Bilanz 1995"
```

Sie fragen sich vielleicht, welcher Unterschied zwischen der „Worksheets-Auflistung und der „Worksheets“-Methode besteht, die die „Worksheets“-Auflistung zurückgibt. Vergessen Sie dieses Problem! Wenn Sie mit Eigenschaften oder Methoden arbeiten, die Objekte, dann behandeln Sie sie genau so , würden Sie mit einem Objekt arbeiten.

© by Josef Broukal, Vienna, Austria

Objekt „Application“ („Anwendung“)

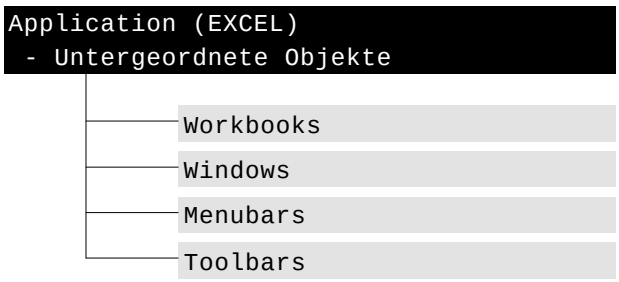


Abbildung 18: EXCEL, „die Mutter aller Objekte“ und die wichtigsten bei Arbeitsbeginn sichtbaren „Container–Objekte“

CONTAINER–OBJEKTE

Die vier in der Abbildung aufgezählten Objekte sind genaugenommen Objekt–Auflistungen oder Container–Objekte. Wollen wir eines der im Container–Objekt „Workbooks“ enthaltenen Objekte ansprechen, müssen wir dies z.B. so tun:

```
Workbooks("Jänner.XLS")
```

oder

```
Workbooks.Item("Jänner.XLS")
```

AKTIVE OBJEKTE

Diese Schreibweise ist natürliches etwas langwierig. Objekte, die gerade im Zugriff (sichtbar, aktiv) sind, können aber einfacher angesprochen werden.

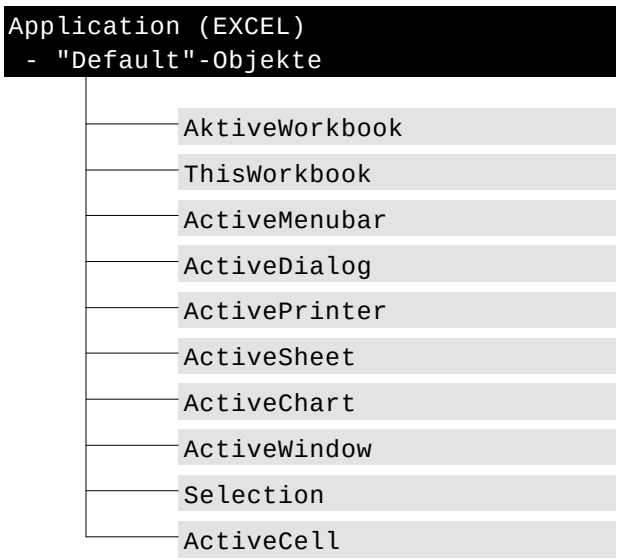


Abbildung 19: EXCEL–Objekte, die gerade sichtbar (ausgewählt) sind, können ohne Namensangabe angesprochen werden.

© by Josef Broukal, Vienna, Austria

ActiveWorkbook Arbeitsmappe, die gerade im Vordergrund liegt (aktiv ist).
ThisWorkbook Arbeitsmappe, deren Code gerade abgearbeitet wird.
ActiveCell Zelle im aktiven Fenster, in der der Cursor steht.
Selection In Tabellenblättern: markierte Zellen, auch Mehrfachauswahl. Bei Fehlen einer Markierung ident mit aktiver Zelle. In Diagrammen oder Dialogblättern: markierte Elemente.

Das „Application"–Objekt ist vielfältig steuerbar. Es liefert Informationen und läßt Werte setzen.

UMGEBUNGS–INFORMATIONEN

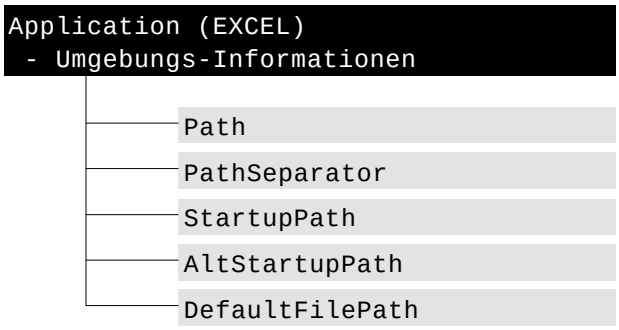


Abbildung 20: Einige Umgebungs–Informationen des „Application"-Objekts

BETRIEBSZUSTANDS–INFORMATIONEN

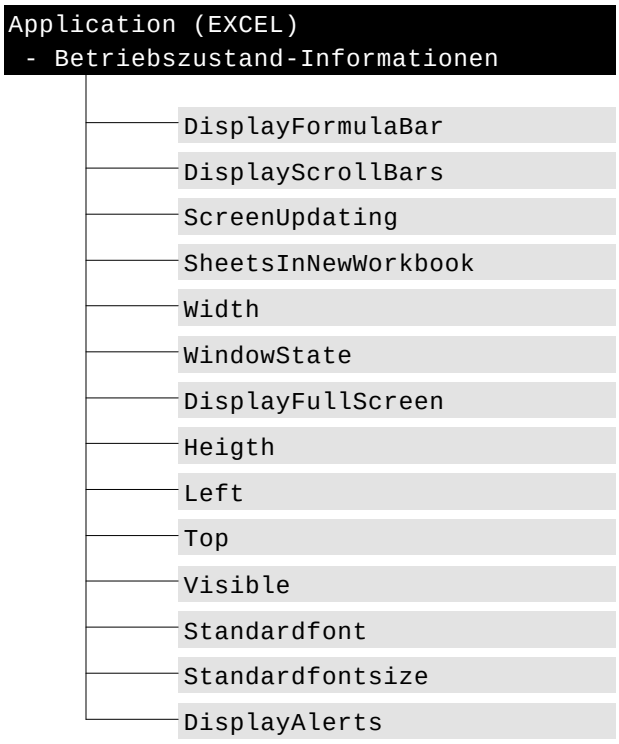


Abbildung 21: Mit diesen Kennworten werden Eigenschaften des „Application"-Objekts

erfragt und gesetzt.

© by Josef Broukal, Vienna, Austria

Container-Objekt „Workbooks“ („Arbeitsmappenliste“)

Dieses Container-Objekt müssen wir immer dann ansprechen, wenn wir an der Gesamtheit der Arbeitsmappen etwas ändern wollen: eine neue Arbeitsmappe anlegen oder öffnen, **alle** Arbeitsmappen auf einmal schließen.

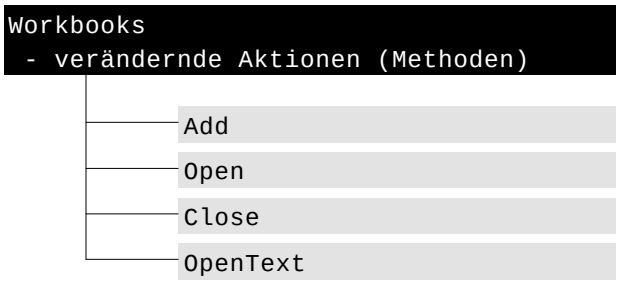


Abbildung 22: Methoden des Container-Objekts „Workbooks“

Objekt „Workbook“ („Arbeitsmappe“)

Wird grundsätzlich als Teil der „Workbooks“-Auflistung angesprochen, ausgenommen bei „ThisWorkbook“ und „ActiveWorkbook“.

Dieses Objekt enthält wieder eine Reihe von Objekten:

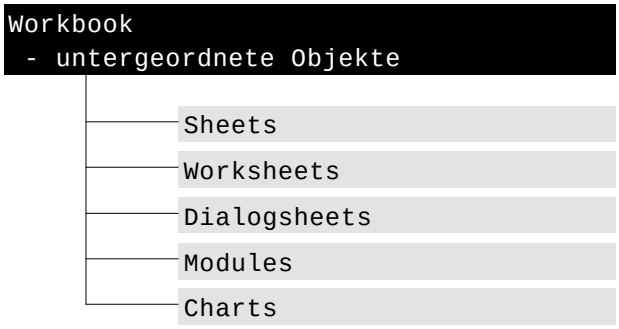


Abbildung 23: Die unterhalb der Arbeitsmappe liegenden Objekte.

„Worksheets“ faßt alle Blätter zusammen, die anderen Container–Objekte nur Blätter eines Blatt–Typs.

Container-Objekt „Windows“ („Fensterliste“)

Dieses Container–Objekt enthält alle Fenster—bedenken Sie bitte immer, daß Sie in EXCEL alle Arbeitsmappen auf dem Bildschirm immer nur durch ein oder mehrere Fenster sehen, nie die Arbeitsmappen selbst. Alles, was das Erscheinungsbild der Arbeitsmappen auf dem Schirm betrifft, muß also über das „Windows“-Container-Objekt oder über Eigenschaften eines Fenster geregelt werden.

© by Josef Broukal, Vienna, Austria

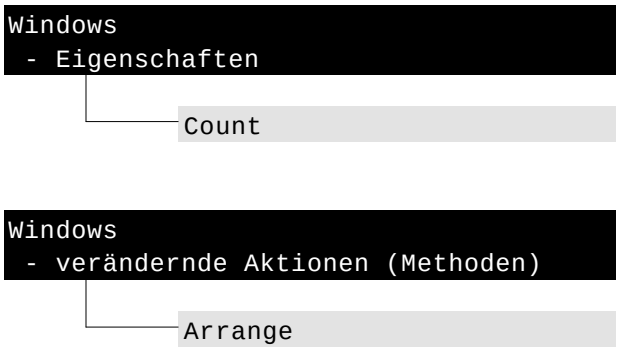


Abbildung 24: Die wichtigste Eigenschaft und Methode des „Windows“–Objekts

© by Josef Broukal, Vienna, Austria

Objekt „Window“ (Fenster“)

Fenster–Objekte regeln über ihre Eigenschaften alles, was Sie im Menü „Fenster“ einstellen können; darüber hinaus auch noch vieles, daß im Register „EXTRAS, Optionen“ eingestellt werden kann:

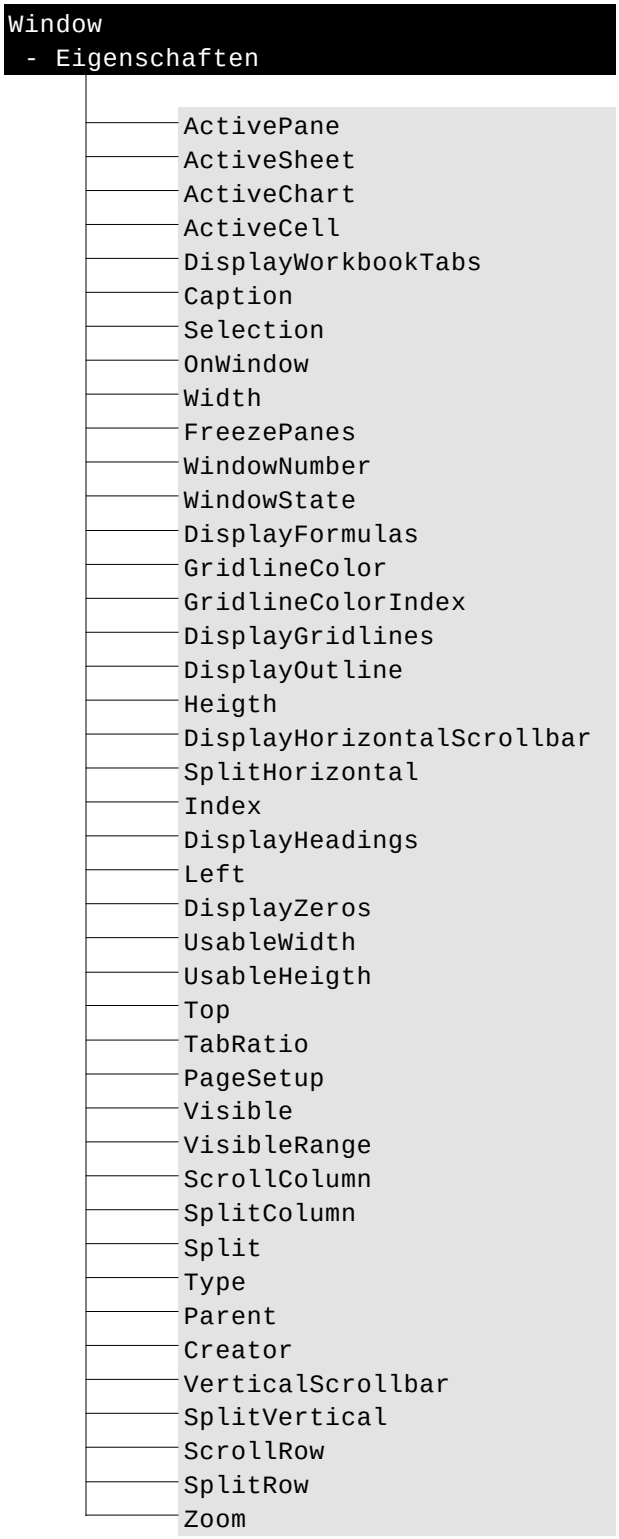


Abbildung 25: Die Eigenschaften des „Window“–Objekts. Die Liste ist deshalb so lang, weil fast jede Fenster-Eigenschaft wichtig für das Arbeiten mit EXCEL ist!

© by Josef Broukal, Vienna, Austria

Auch die Methoden des „Window“–Objekts sind wichtig:

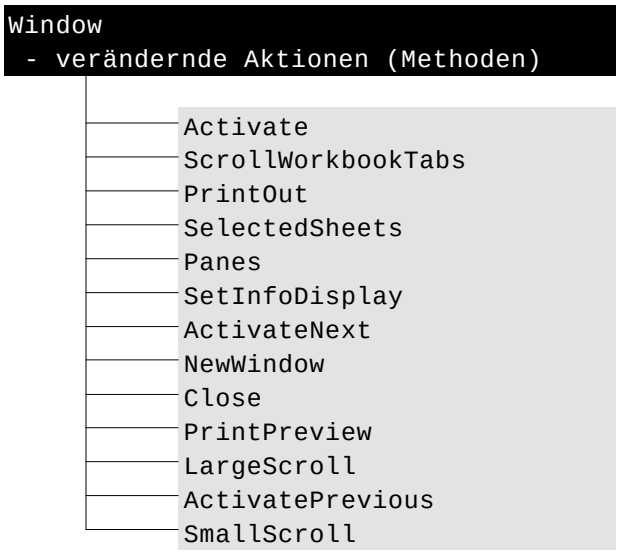


Abbildung 26: Methoden des „Window“–Objektes

© by Josef Broukal, Vienna, Austria

Objekte 2: Zellen und Zellbereiche

Bereiche aus mehreren Zellen

Nicht die einzelne Zelle, sondern der Zellbereich ist das kleinste Objekt in einem Tabellenblatt (obwohl natürlich ein Zellbereich auch nur eine Zelle enthalten kann). Dieses Bereichs–Objekt hat wieder eine Reihe erfrag- und setzbarer Eigenschaften:

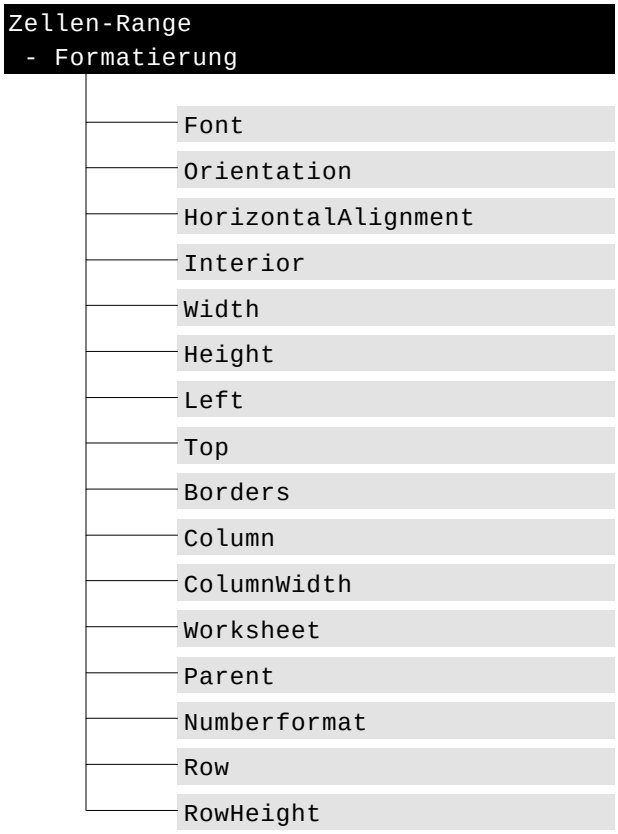


Abbildung 27: Die wichtigsten Eigenschaften eines Zellbereiches.

Das Schriftart-Objekt eines Bereiches wird über die Schriftart-Eigenschaft eines Bereichs-Objektes definiert und dann über die folgenden Eigenschaften näher definiert:

© by Josef Broukal, Vienna, Austria

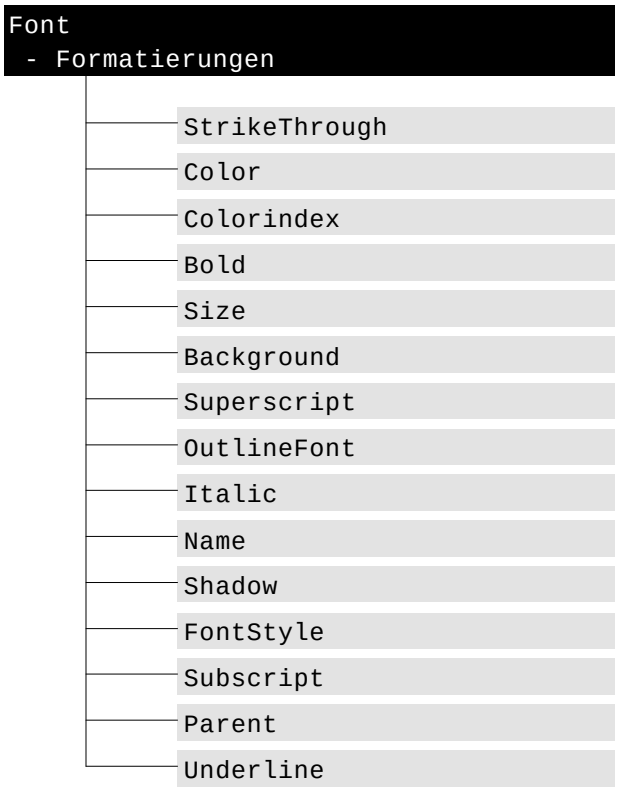


Abbildung 28: Eigenschaften des Schriftart-Objektes.

Weiters können Zellbereiche mit Rahmen (innen wie außen) versehen werden. Das Rahmen-Objekt wird mit Hilfe der „RahmenListe“-Methode des Bereichs-Objektes erfragt und dann über die folgenden Eigenschaften formatiert:

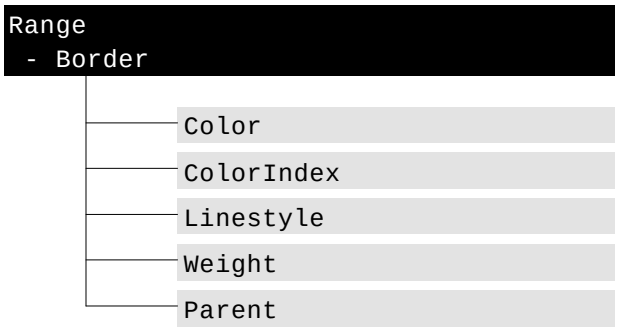


Abbildung 29: Eigenschaften des Rahmen-Objektes.

Schließlich ist auch der Innenbereich jeder Zelle eines Bereiches veränderbar. Das Innenbereichs-Objekt wird über die Innenbereich-Eigenschaft des Bereichs-Objektes ermittelt und dann mit den folgenden Eigenschaften in seinem Aussehen definiert:

© by Josef Broukal, Vienna, Austria

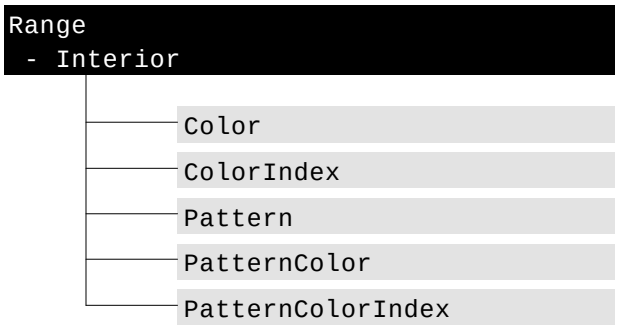


Abbildung 30: Eigenschaften des Innenbereichs eines Zellbereiches.

Schließlich kann auch der ganze Bereich manipuliert werden.

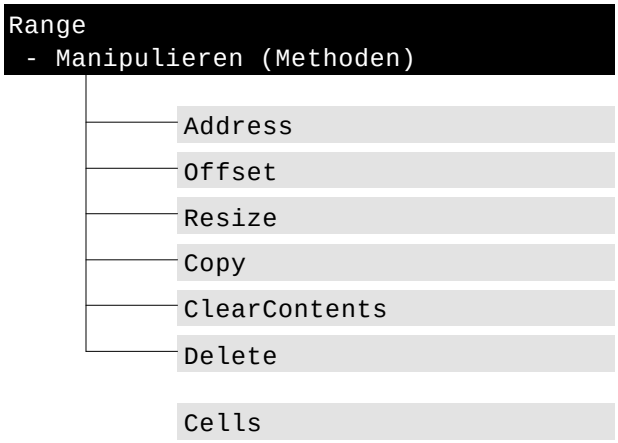


Abbildung 31: Informationen über einen Bereich; Änderung des Bereichs - dazu dienen diese Methoden.

Sonderfall: Bereich umfaßt nur eine Zelle

Ein solcher Bereich hat zusätzliche Eigenschaften:

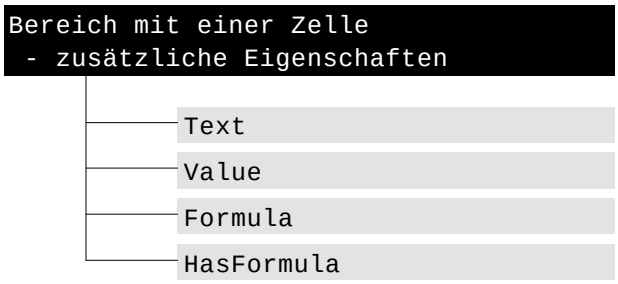


Abbildung 32: Eigenschaften von Zellbereichen, die aus einer einzelnen Zelle bestehen.

Jede Zelle eines EXCEL–Arbeitsblattes kann auf vielfältige Weise beschrieben werden:

- Welchen Wert hat sie
- In welcher Zeile und Spalte steht sie
- Wie ist sie formatiert, usw.

© by Josef Broukal, Vienna, Austria

Diese Informationen über eine Zelle werden „Eigenschaften“ genannt.

Die Eigenschaften einer Zelle können durch VBA verändert werden:

- Die Zelle erhält einen Wert zugewiesen
- Sie erhält ein Zahlenformat, eine Schriftart, einen Rahmen

Auch mit „Methoden“ kann eine Zelle bearbeitet werden:

- Wir können ihre Adresse feststellen
- Wir können sie ausschneiden und an anderer Stelle wieder einfügen
- Wir können ihren Inhalt löschen, usw.

Eine besondere Zelle ist die, in der der Cursor steht—sie ist die „aktive Zelle“ und kann auch so angesprochen werden, als „ActiveCell“.

Wie kommen wir zur Zelle B5 des Arbeitsblattes „Tabelle1“?

Jede Zelle eines Tabellenblattes ist Teil eines Objektes—eines „Bereichs–Objektes“. Das Bereichs–Objekt (englisch „Range“) wieder ist in „Container–Objekten“ eingebettet—Tabellenblatt, Arbeitsmappe, Anwendung, also EXCEL 5.0. Je „näher“ wir dem Bereichs–Objekt sind, um so einfacher können wir es ansprechen:

Ist das Tabellenblatt „Tabelle1“ aktiv (es ist im aktiven Fenster sichtbar), dann genügen Zellreferenz und „Select“–Methode:

```
[B5].Select
Cells(5, 2).Select
```

um diese Zelle zur aktiven Zelle zu machen. (Nach dem Ende des VBA–Programmes, das diesen Befehl enthält, würde diese Zelle mit dem Cursor–Rahmen versehen sein.)

Ist ein anderes Blatt als „Tabelle1“ sichtbar, dann müssen Sie bei der Zellauswahl auch den Blattnamen angeben:

```
[Tabelle1!B5].Select
worksheets(„Tabelle1“).Cells(5, 2).Select
```

Ist sogar eine andere Arbeitsmappe aktiv, dann müssen Sie zunächst die Mappe aktivieren, in der „Tabelle1“ enthalten ist:

```
Workbooks("Tutor1.xls").Activate
[Tabelle1!B5].Select
```

Da unter VBA eine Zelle eine „Range“ („Bereich“) ist, können wir auch schreiben:

```
Range("B5").Select

Worksheets("Tabelle4“).Range("B5").Select

Workbooks(„Tutor1.XLS“).Activate
Worksheets("Tabelle4“).Range("B5").Select
```

© by Josef Broukal, Vienna, Austria

Eigenschaften von Zelle B5 setzen und erfragen

[A1]–FORMAT

```
[B5].Select
[B5].Value = "Test"
[B5].Font.Size = 14
[B5].Font.Bold = Wahr

MsgBox "Schriftart in Zelle B5 ist: " + _
    [B5].Font.Name
MsgBox "Inhalt von Zelle B5 ist: " + _
    [B5].Value
```

ALS BEREICHS-OBJEKT

```
Range("B5").Value = "Test2"
Range("B5").Font.Size = 12
Range("B5").Font.Bold = False

"Schriftart in Zelle B5 ist: " + _
    Range("B5").Font.Name
MsgBox "Inhalt von Zelle B5 ist: " + _
    Range("B5").Value
```

Eigenschaften mehrerer Zellen setzen und erfragen

[A1]–FORMAT

```
[A1].Select
[A3:B6].Value = "Test"
[A3:B6].Font.Size = 14
[A3:B6].Font.Bold = True
```

Beim Erfragen von Eigenschaften mehrerer Zellen ist Vorsicht geboten. Bei der Schriftart geht das noch—falls jede Zelle die selbe Schriftart hat. Bei der Eigenschaft „Value“ streikt EXCEL und gibt eine Fehlermeldung aus.

ALS BEREICHS–OBJEKT

Folgende Arbeitsschritte sind dabei zu setzen:

Bereich definieren Bleiben wir wieder bei unserem Bereich, den Zellen A3 bis B6 in „Tabelle2“. Wir schreiben diesfalls

```
Sub Hauptprogramm_31()

Set MeinBereich = ActiveWorkbook.Sheets("Tabelle2").Range("A3:B6")
MsgBox MeinBereich.Address(external:= True)

End Sub
```

lassen das VBA–Programm laufen und erhalten im Meldungsdialog die Zelladresse.

Bereich mit Wert belegen Dazu müssen wir den Zellen des Bereiches den Wert zuweisen (Objekt–Gurus würden sagen: „Die Wert–Eigenschaft des Bereichs–Objektes setzen“.)

© by Josef Broukal, Vienna, Austria

```
Sub Hauptprogramm_32()

Set MeinBereich = ActiveWorkbook.Sheets("Tabelle2").Range("A3:B6")
MeinBereich.Value = 12

End Sub
```

Werte aus allen Zellen des Bereiches auslesen Dies geht am einfachsten so, daß wir aus jeder einzelnen Zelle des Bereiches ein Objekt bilden, dessen Wert wir erfragen und anzeigen.

```
Sub Hauptprogramm_33()

Set Meinbereich = ActiveWorkbook.Sheets("Tabelle2").Range("A3:B6")
For Each EineZelle In Meinbereich.Cells
    MsgBox EineZelle.Address
    MsgBox EineZelle.Value
Next

End Sub
```

© by Josef Broukal, Vienna, Austria

Zum Start zwei kurze Beispiele…

Automatisch Zoomen

Wenn Ihre Tabelle zu groß geraten ist, um bei normaler Schiftgröße auf dem Schirm ganz sichtbar zu sein, heißt es, Text und Zellen verkleinern. EXCEL bietet dazu im Menü „Ansicht“ den Befehl „Zoom“ an und alternativ, in der „Standard“-Symbolleiste, das Werkzeug „Zoom“. Beide erfordern einige Tastendrucke bzw. Klicke mit der Maus.

Ein kleines VBA-Programm, „Automatic Zoom“, ist da weit bequemer. Es funktioniert so:

- Ist in der aktiven Tabelle nur eine Zelle ausgewählt (und in irgendeiner steht der Zellcursor immer!), interpretiert das Programm, sobald es gestartet wurde, das als Auftrag, die Tabelle in normaler Größe (Zoom 100%) darzustellen.
- Ist mehr als eine Zelle markiert, vergrößert oder verkleinert „Automatic Zoom“ die Darstellung so weit, daß alle markierten Zellen auf dem Bildschirm sichtbar werden.

Hier ist der Code. Schreiben Sie ihn in ein neues Makroblatt in einer neuen Arbeitsmappe.

```
Sub AutomaticZoom()

If TypeName(ActiveSheet) <> "Worksheet" Then
  Beep
  Exit Sub
End If

If Selection.Cells.Count > 1 Then
  ActiveWindow.Zoom = True
Else
  ActiveWindow.Zoom = 100
End If

End Sub
```

Das Programm kann mit dem Makro-Rekorder nicht aufgezeichnet werden. Es enthält nämlich Anweisungen, die man dem Rekorder nicht vorzeigen kann: Die mit „If“, „Else“ und „End If“ eingefaßten Codezeilen, die je nachdem, wie viele Zellen in einer Tabelle markiert sind, einmal abgearbeitet werden und ein anderes Mal wieder nicht.

Müßte „AutomaticZoom“ jedesmal über das Menü „Extras“ und den Befehl „Makro…“ aufgerufen werden, dann brächte es wohl keine Arbeitsersparnis. Erst mit einem Tastatur-Shortcut ist es schnell zur Hand.

Und den wollen wir jetzt einrichten. Wir wählen die Tastenkombination „Strg“+“Umschalt“+“Z“. Wie ordnen wir sie „AutomaticZoom“ zu? Über den Befehl „Makro“ im Menü „Extras“. Dort sind alle VBA-Programme aufgelistet, die gerade verfügbar sind. Klicken Sie einmal auf „AutomaticZoom“ und dann auf den Schaltknopf „Optionen…“.

© by Josef Broukal, Vienna, Austria

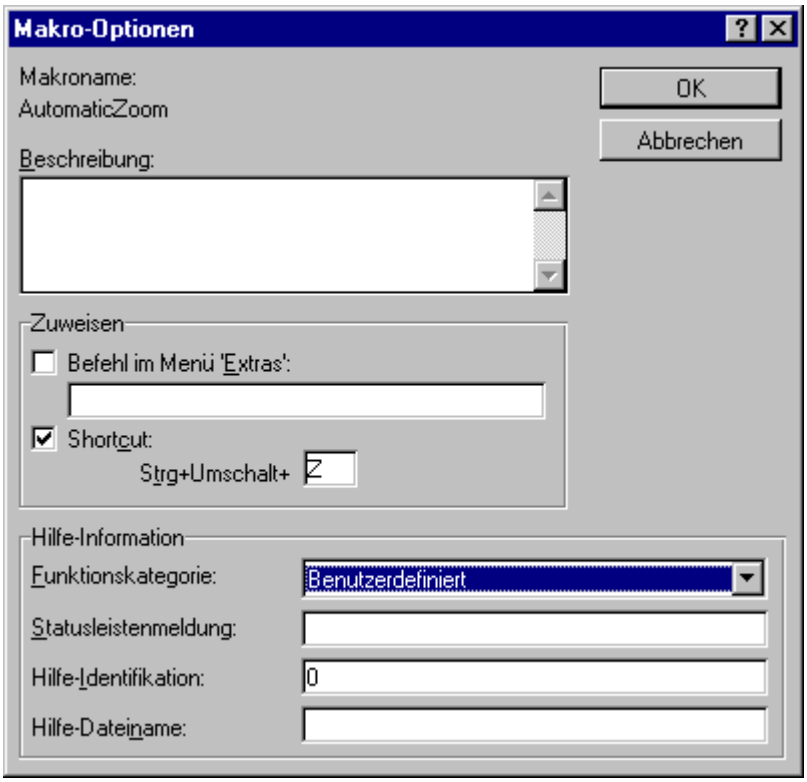


Abbildung 33: **Im Dialog „Makro-Optionen“ werden die Tasten „Strg“+“Umschalt“+“Z“ dem Programm „AutomaticZoom“ zugeweisen.**

Paßt. Schliessen Sie den Dialog wieder. „AutomaticZoom“ ist betriebsbereit und wird es immer sein, wenn die Arbeitsmappe, in der sein Code steht, geladen ist. Schön wäre es, würde die Mappe beim Starten von EXCEL von selbst geladen. Während der Arbeit mit EXCEL sollte sie allerdings unsichtbar sein.

Und das geht so: Speichern Sie die Mappe unter dem Namen „ZOOM.XLS“ auf Ihrer Festplatte in das Verzeichnis „XLStart“, das Sie unterhalb des Verzeichnisses „EXCEL“ finden werden. (Beim Installieren von EXCEL wurde dieser Ordner mit angelegt.)

Machen Sie „AutomaticZoom“ dann über das Menü „Fenster“, Befehl „Ausblenden“ unsichtbar. Rufen Sie das Programm dann noch einmal auf. EXCEL sieht das so, als hätten Sie das Programm verändert und wird Sie beim Schließen fragen, ob „ZOOM.XLS“ noch einmal gespeichert werden soll. Tun Sie das—erst jetzt wird die Information, daß die Mappe verborgen ist, mit abgespeichert.

Spalten so breit wie ihr Inhalt machen

Für Text-Informationen ist die vorgegebene Spaltenbreite in den EXCEL-Tabellen oft zu klein. Sie über das Menü „Format“, den Befehl „Spalte“ und den Unterbefehl „Optimale Breite“ so breit zu machen, wie es der Text braucht, ist möglich, aber mühsam. Bequemer geht es mit einem VBA-Programm. Weil Sie dieses Programm wahrscheinlich ebensowenig werden missen wollen wie „AutomaticZoom“, sollten Sie den Code in die selbe Mappe schreiben. Und zwar in ein neues Modulblatt.

© by Josef Broukal, Vienna, Austria

```
Sub OptimaleBreiteEinstellen()
If TypeName(ActiveSheet) <> "Worksheet" Then
  Beep
  Exit Sub
End If
Selection.Columns.AutoFit
End Sub
```

Die „Selection…“-Anweisung habe ich übrigens mit dem Makro-Rekorder eingefügt. Das geht schneller als das Blättern in der Online-Hilfe auf der Suche nach dem englischen Namen für „Optimale Breite“…

Dieses Programm wird wohl weniger oft gebraucht werden als „AutomaticZoom“. Es soll deshalb ins Menü „Format“ gleich unter den Eintrag „Spalte“ gestellt werden. Das geht—bei sichtbarem Modulblatt—am einfachsten über den Menü-Editor, dessen Symbol in der Symbolleiste „Visual Basic“ angeboten ist.

© by Josef Broukal, Vienna, Austria

Steuern des Programmablaufs

In diesem Kapitel lernen Sie:

- Bedingtes Ausführen von Codeabschnitten (Entscheidungsstrukturen)
- Mehrfaches Ausführen von Codeabschnitten (Kontrollstrukturen)
- Kontrollstrukturen innerhalb anderer Kontrollstrukturen
- Verlassen von Kontrollstrukturen und Prozeduren

Wir haben bis jetzt im Großen und Ganzen nur Programme kennengelernt, die wie ein Uhrwerk ablaufen—vom Start zum Ende, und das nur einmal. In diesem Kapitel wollen wir Entscheidungs- und Kontrollstrukturen kennenlernen. Sie steuern ein Programm in Abhängigkeit davon, welche Ereignisse während des Programmlaufes auftreten.[1]

Bedingtes Ausführen von Codeabschnitten

VBA–Programme können in ihnen enthaltene Bedingungen überprüfen und je nach Ergebnis der Prüfung unterschiedliche Operationen ausführen.

If…Then („Wenn…Dann“)

Diese Struktur führt die in ihr stehenden Anweisungen nur dann aus, wenn die Prüfung ein logisches „Wahr“ (englisch „True“) ergibt, wie im folgenden Beispiel:

```
Sub WennDann_1()
If EinDatum <> Now Then EinDatum = Now
End Sub
```

oder

```
Sub WennDann_2()

If EinDatum < Now Then
    EinDatum = Now
    MsgBox "Das heutige Datum ist der " & EinDatum
End If

End Sub
```

Was ist der Unterschied? Bei der zweiten Version können Sie nach „If“ beliebig viele Anweisungen schreiben, bei der ersten Version nur eine. Beachten Sie, daß die zweite Version am Ende mit der Anweisung „End If“ abgeschlossen werden muß.

---

[1] Die Beispiele und Erklärungen in diesem Kapitel sind teilweise der Online–Hilfe für VBA und dem mit EXCEL 5.0 mit ausgelieferten „Visual Basic Benutzerhandbuch“ entnommen.

© by Josef Broukal, Vienna, Austria

If…Then…ElseIf…Else…End If(„Wenn…Dann…Sonst“)

In dieser Struktur kann eine beliebige Anzahl verschiedener Bedingungen festgelegt werden. Die Bedingungen werden der Reihe nach geprüft. Trifft eine zu, werden die hinter ihr stehenden Anweisungen ausgeführt. Die folgenden Bedingungen werden nicht mehr geprüft.

Syntax

If Bedingung **Then**
  Anweisungen…
ElseIf AndereBedingung **Then**
  Anweisungen…
ElseIf WiederAndereBedingung **Then**
  Anweisungen…
Else
  Anweisungen…
End If

„Bedingung“ ist immer ein logischer Vergleich, der den Wert „True“ ergeben muß. Wird keine Bedingung erfüllt, werden die Anweisungen unterhalb von „Else“ ausgeführt (wenn Sie dorthin keine Anweisungen setzen, funktioniert die Sache auch).

Beispiel

```
Sub WennDann_1()

If EinDatum = Now Then EinDatum = Now

End Sub


Sub WennDann_2()

If EinDatum < Now Then
    EinDatum = Now
    MsgBox "Das heutige Date ist der " & EinDatum
End If

End Sub
```

Hinweis Wenn „Bedingung“ zwei Teilbedingungen enthält wie in

```
If (IsMissing(A) Or (A > 9))
```

werden beide Teilbedingungen geprüft, auch wenn schon die erste Teilbedingung „True“ ergibt. Es müssen also immer beide Bedingungen prüfbar sein. Gibt es in unserem Beispiel die Variable „A“ nicht, dann ist die Bedingung „IsMissing (A)“ erfüllt; dennoch überprüft VBA auch die Bedingung „A > 9“—und unterbricht das Programm, weil es „A“ nicht findet.

Ist es beim Erstellen des VBA–Programmes nicht klar, ob die Variable „A“ zur Laufzeit vorhanden ist, müssen Sie also schreiben:

© by Josef Broukal, Vienna, Austria

```
If IsMissing(A) Then
    Anweisungen…
ElseIf A > 9 Then
    Anweisungen…
End If
```

Wenn Sie mehr als ein, zwei Anweisungen angeben, ist es sinnvoller, sie legen die Anweisungen in eine eigene Prozedur:

```
If IsMissing(A) Then
    MeineProzedur
ElseIf A > 9 Then
    MeineProzedur
End If

Sub MeineProzedur()
    Anweisungen…
End Sub
```

Select Case

Im Gegensatz zu „If…Then…Else“, wo Sie beliebig viele, sich auf ganz verschiedene Variable und Programmzustände beziehende Bedingungen angeben können, arbeitet „Select Case“ mit nur einem einzigen „Testausdruck“, dessen Wert Sie allerdings in beliebig vielen „Case“–Anweisungen untersuchen können. Sie können den Testausdruck auch einer Funktion zuweisen.

Syntax

Select Case Testausdruck
  Case Ausdrucksliste1
    Anweisungen…
  Case Ausdrucksliste2
    Anweisungen…
  Case Else
    Anweisungen…
End Select

© by Josef Broukal, Vienna, Austria

Beispiel

```
Sub PrüfeFall_1()

Number = InputBox("Gib eine Zahl ein:") ' Variable initialisieren.
Select Case Number ' Zahl auswerten.
Case 1 To 5    ' Zahl im Bereich von 1 bis 5.
    TestZeichenF = "im Bereich von 1 To 5"
Case 6, 7, 8, 9, 10 ' Zahl im Bereich von 6 bis 10.
    TestZeichenF = "im Bereich von 6 To 10"
Case MeineFunktion(Number)
    TestZeichenF = "Funktion <<MeineFunktion>> ausgeführt"
Case Else  ' andere Werte.
    TestZeichenF = "nicht im Bereich von 1 bis 10"
End Select
MsgBox "Die Zahl liegt " & TestZeichenF

End Sub

Function MeineFunktion(x)

MsgBox "Bin in <<MeineFunktion>>"

End Function
```

Mehrfaches Ausführen desselben Codes mit Programmschleifen

Dies geschieht mit Hilfe von Schleifenstrukturen, von denen VBA drei kennt. Die Anweisungen zwischen dem Kopf und dem Fuß der Schleife werden solange ausgeführt, bis die „Schleifenbedingung“ nicht mehr zutrifft.

Do While…Loop

Wiederholt einen Block mit Anweisungen, solange eine Bedingung „Wahr“ (englisch „True“) ist, oder bis die Bedingung „True“ wird. Setzen Sie diese Schleife ein, wenn Sie beim Programmieren nicht wissen können, wie oft die Anweisungen in der Schleife zur Laufzeit des VBA–Programmes ausgeführt werden müssen.

Syntax 1

Do [{While | Until} Bedingung**]**
  [Anweisungen]
  [Exit Do]
  [Anweisungen]
Loop

Syntax 2

Do
  [Anweisungen]
  [Exit Do]
  [Anweisungen]
Loop [{While | Until}Bedingung**]**

Elemente

© by Josef Broukal, Vienna, Austria

Bedingung ein Ausdruck, der entweder Wahr oder Falsch ist.

Anweisungen eine oder mehrere Anweisungen, die wiederholt werden, bis Bedingung Wahr wird.

Exit Do verläßt die Kontrollstuktur vom Typ Do…Loop „vorzeitig“, also wenn „Bedingung“ noch zutrifft. Exit Do läßt sich an beliebiger Stelle und beliebig oft in einer solchen Struktur verwenden. Diese Anweisung wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel If…Then) eingesetzt und setzt die Ausführung mit der ersten Anweisung im Anschluß an die Schleife fort. In verschachtelten Do…Loop übergibt Exit Do die Steuerung an die Schleife der nächst-niedrigeren Verschachtelungsstufe.

KOPFGESTEUERTE „DURCHLAUFE“–SCHLEIFE

Bei dieser Form der „Do“–Schleife wird „Bedingung“ schon geprüft, bevor die Schleife auch nur ein einziges Mal durchlaufen worden ist. Ist „Bedingung“ schon vor dem ersten Durchlauf „False“, werden die Anweisungen kein einziges Mal ausgeführt; dasProgramm setzt in der Zeile nach „Loop“ fort.

Beispiel

```
Sub DurchlaufeKopf_1()
LangeZeichenfolge = "Ich bin eine lange Zeichenfolge"
KurzeZeichenfolge = "i"
MsgBox ZeichenfolgenAnzahl(LangeZeichenfolge, _
    KurzeZeichenfolge)
End Sub

Function ZeichenfolgenAnzahl(LangeZnF, SuchZnF)
Position = 1
Do While InStr(Position, LangeZnF, SuchZnF)
    Position = InStr(Position, LangeZnF, SuchZnF) + 1
    Count = Count + 1
Loop
ZeichenfolgenAnzahl = Count
End Function
```

Die hier beschriebene Funktion sucht in einer Zeichenkette („LangeZnF“) nach einer zweiten Zeichenkette („SuchZnF“) und zählt, wie oft diese gefunden wurde. Wird die zweite Zeichenkette kein einziges Mal gefunden, dann ergibt die Bedingung „InStr(Position; LangeZnF; SuchZnF)“ den Wert 0. Dies übersetzt VBA in „False“; die Schleife wird kein einziges Mal durchlaufen. Jede andere Zahl wird mit „True“ übersetzt, die Schleife wird durchlaufen.

FUSSGESTEUERTE „DO“–SCHLEIFE

Bei dieser Form der Schleife werden die in der Schleife gegebenen Anweisungen zunächst ausgeführt, erst dann wird die am Ende der Schleife definierte Schleifenbedingung geprüft. Diese Form der „Durchlaufe“–Schleife verwenden Sie immer dann, wenn Sie sicher sind, daß die Anweisungen zumindest einmal ausgeführt werden müssen

© by Josef Broukal, Vienna, Austria

```
Sub Durchlaufe_Fuß()
Do
BenutzerdatenVerarbeiten
Antwort = MsgBox("Weitere Daten verarbeiten?", vbYesNo)
Loop While Antwort = vbYes
End Sub

Sub BenutzerdatenVerarbeiten()
MsgBox "Bin in BenutzerdatenVerarbeiten"
End Sub
```

AUSFÜHREN, SOLANGE EINE BEDINGUNG FALSCH IST

Die beiden gerade gezeigten „Do“–Schleifen werden solange abgearbeitet, solange die Schleifenbedingung „True“ ergibt. Mit Hilfe des Befehlswortes „BisWahr“ können Sie aber auch „Durchlaufe“–Schleifen bilden, die solange durchlaufen werden, solange die Bedingung „Falsch“ ergibt.

```
Sub Durchlaufe_Fuß()
Do
BenutzerdatenVerarbeiten
Antwort = MsgBox("Weitere Daten verarbeiten?", vbYesNo)
Loop Until Antwort = vbNo
End Sub

Sub BenutzerdatenVerarbeiten()
MsgBox "Bin in BenutzerdatenVerarbeiten"
End Sub
```

For…Next

Diese Art der Schleife sollten Sie immer dann verwenden, wenn die Zahl der Durchläufe zumindest während des Programmlaufes vor Eintritt in den Schleifencode feststeht. Sie wiederholt solange eine Reihe von Anweisungen, bis der Schleifenzähler einen bestimmten Wert erreicht.

Syntax

For Zähler = Anfang **To** Ende [**Step** Schritt]
  [Anweisungen]
  [**Exit For**]
  [Anweisungen]
Next [Zähler]

Elemente

Die For...Next-Anweisung verwendet die folgenden Argumente:

Zähler Numerische Variable, die als Schleifenzähler dient. Ein Element eines Datenfelds oder eines benutzerdefinierten Typs ist an dieser Stelle nicht zulässig.

Anfang Startwert von Zähler.

Ende Endwert von Zähler.

© by Josef Broukal, Vienna, Austria

Schritt Schrittweite, um die Zähler bei jedem Schleifendurchlauf verändert wird. Diese Angabe ist optional; wenn Sie sie weglassen, nimmt VBA als Schrittweite +1 an. **Bitte beachten Sie, daß bei Schrittweiten, die Nachkommastellen enthalten, Rundungsfehler auftreten können!**

Anweisungen Eine oder mehrere Anweisungen zwischen For und Next, die mehrmals ausgeführt werden.

Negative Schrittweite

Wenn „Ende“ größer ist als „Anfang“, dann müssen Sie „Schritt“ angeben. Die Schleife wird sonst kein einziges Mal durchlaufen!

Anmerkungen

Nachdem im ersten Schleifendurchlauf alle Anweisungen ausgeführt wurden, addiert das Programm Schritt zum Wert von Zähler hinzu und führt denselben Test durch wie zu Beginn des ersten Durchlaufs. Wenn der Endwert noch nicht überschritten (bzw. bei negativem Schritt unterschritten) wurde, führt das Programm die Schleife erneut aus, ansonsten setzt es die Ausführung mit der Anweisung fort, die auf die Next-Anweisung folgt.

Beachten Sie, daß Zähler nach dem Verlassen der Schleife nicht den Wert von Ende hat, sondern von Ende+Schritt beziehungsweise Ende-Schritt.

Schleife mit „Exit For“ vorzeitig verlassen

Exit For führt zum sofortigen Verlassen der Schleife, auch wenn „Ende“ noch nicht erreicht worden ist. Es läßt sich an beliebigen Stellen und beliebig oft in der Schleife verwenden. Exit For wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel If…Then) eingesetzt und weist das Programm an, die Ausführung mit der ersten Anweisung hinter Next fortzusetzen.

Mehrere Schleifen ineinander

Sie können For…Next-Schleifen verschachteln, indem Sie eine dieser Schleifen innerhalb einer anderen verwenden. Die Zähler für alle Schleifen müssen dabei jedoch durch verschiedene Variablennamen dargestellt werden, wie im folgenden Beispiel:

```
For I = 1 To 10
  For J = 1 To 10
    For K = 1 To 10
      . . .
    Next K
  Next J
Next I
```

Hinweis Das Argument Element (z.B. „Next K“) in der Next-Anweisung ist nicht zwingend notwendig. Die Ausführung wird mit einem Element genauso fortgesetzt wie ohne Element. Eine Next–Anweisung vor der zugehörigen For-Anweisung führt zu einem Fehler.

For Each…Next

Eine hochinteressante Schleife, so richtig auf VBA zugeschniten. Sie arbeitet nämlich nicht mit Wahrheitswerten oder Zählern, sondern mit Objekten. Genauer gesagt, mit all jenen Objekten, die Auflistungen enthalten. Wieviele Elemente die Auflistung hat, brauchen Sie gar nicht zu wissen.

© by Josef Broukal, Vienna, Austria

Syntax

For Each Element **In** Gruppe
[Anweisungen]
[Exit For]
[Anweisungen]
Next [Element]

Die Anweisung For Each...Next verwendet die folgenden Argumente:

Argument	Beschreibung
Element	Variable zum Durchlauf durch die Elemente einer Auflistung oder eines Datenfeldes. Bei Auflistungen sind für Element nur Variant-Variablen, allgemeine Variablen vom Typ Objekt oder spezielle Objektvariablen für die OLE-Automatisierung zulässig. Bei Datenfeldern sind für Element nur Variant-Variablen zulässig.
Gruppe	Name einer Auflistung von Objekten oder eines Datenfeldes (abgesehen von Datenfeldern mit einem benutzerdefinierten Typ).
Anweisungen	Eine oder mehrere Anweisungen, die für jedes Element in Gruppe ausgeführt werden.

Wichtig Wenn Sie For Each...Next mit Datenfeldern verwenden, können Sie den Wert der Datenfeldelemente, die durch die Kontrollvariable Element bestimmt werden, nur lesen. Es ist nicht möglich, den Wert zu ändern, indem Sie Element einen Wert zuweisen.

Beispiel Angenommen, Sie wollen alle offenen Arbeitsmappen schließen, ausgenommen die eine, die Ihr Programm enthält. Alle Arbeitsmappen sind in dem Container-Objekt „Workbooks“ enhalten—egal, wieviele es sind und wie sie heißen. Wir können dann schreiben:

```
Sub FürAlle_1()
For Each EineMappe In Workbooks
    If EineMappe.Name <> ThisWorkbook.Name Then EineMappe.Close
Next
End Sub
```

Zweites Beispiel: Wir wollen alle Zellen in einem mit der Maus markierten Bereich fett schreiben, deren Wert größer als 1000 ist:

```
Sub FürAlle_2()
For Each EineZelle In Selection
    Wenn EineZelle.Value > 1000 Then
        EineZelle.Font.Bold = True
    End If
Next
End Sub
```

EINSATZ IN DATENFELDERN

For Each…Next können Sie auch verwenden, um Datenfelder zu bearbeiten:

© by Josef Broukal, Vienna, Austria

```
Sub FürAlle_InDatenfeldern_1()
Dim MeinDatenfeld(9) As Variant
For Zähler = 0 To 9
    MeinDatenfeld(Zähler) = Zähler
Next
For Each EinElement In MeinDatenfeld
    MsgBox "Aktuelles Element hat den Wert: " & _
        EinElement
Next
End Sub
```

Falls das Datenfeld mehr als eine Element–Dimension hat:

```
' In einem Datenfeld mit mehr als einer Dimension
' ist die "For Each"-Schleife nicht sinnvoll
Sub FürAlle_InDatenfeldern_2()
Dim MeinDatenfeld(9, 1) As Variant
For Zähler = 0 To 9
    MeinDatenfeld(Zähler, 1) = Zähler
Next
For Each EinElement In MeinDatenfeld
    MsgBox "Aktuelles Element hat den Wert: " & _
        EinElement
Next
End Sub
```

Wenn Sie diesen Code laufen lassen, werden Sie merken, daß die „For Each“–Schleife alle 20 Elemente des Datenfeldes durchläuft. Sie ist also in diesem Fall nicht brauchbar. Verwenden Sie stattdessen eine „For“–Schleife:

```
' Sie verwenden also besser eine "For"-Schleife
Sub Für_InDatenfeldern_1()
Dim MeinDatenfeld(9, 1) As Variant
For Zähler = 0 To 9
    MeinDatenfeld(Zähler, 1) = Zähler
Next
For Schleifenzähler = LBound(MeinDatenfeld) To _
    UBound(MeinDatenfeld)
    MsgBox "Aktuelles Element hat den Wert: " & _
        MeinDatenfeld(Schleifenzähler, 1)
Next
End Sub
```

Schleifen vorzeitig verlassen

Wenn eine Schleife ihren Dienst verrichtet hat, bevor die Schleifenbedingung erfüllt oder der Endstand eines Zählers überschritten worden ist, können Sie das Ausführen der Anweisungen in der Schleife mit Hilfe einer der „Exit“–Anweisungen beenden.

Exit Do
Exit For

brechen jeweils die in ihren Namen enthaltene Schleifenstruktur ab. Für das Verlassen von Prozeduren und Funktionen gibt es ähnliche Anweisungen:

Exit Sub
Exit Function

© by Josef Broukal, Vienna, Austria

Beispiel

Der nachfolgende Code verwendet „Exit“-Anweisungen, um eine For…Next-Schleife, eine Do…Loop-Schleife und eine Sub-Prozedur zu verlassen.

```
Sub ExitAnweisungDemo()
  Do  ' Endlosschleife.
    For I = 1 To 1000  ' 1000 Durchläufe.
      TestNum = Int(Rnd * 1000) ' Zufallszahl generieren.
      Select Case TestNum  ' Zufallszahl auswerten.
        Case 7: Exit For  ' Wenn 7, For...Next beenden.
        Case 29: Exit Do  ' Wenn 29, Do...Loop beenden.
        Case 54: Exit Sub ' Wenn 54, Sub-Prozedur beenden.
      End Select
    Next I
  Loop
End Sub
```

Automatisches Ablaufen von Programmen

Beim Laden einer Arbeitsmappe

Wenn Sie eine Arbeitsmappe laden, dann untersucht EXCEL, ob es in dieser Mappe eine VBA-Prozedur mit dem Namen „auto_open“ gibt. Wird sie gefunden, dann werden die in ihr enthaltenen Anweisungen automatisch abgearbeitet. Gleichermassen werden Anweisungen einer Prozedur mit dem Namen „auto_close“ automatisch beim Schliessen einer Arbeitsmappe durchgeführt.

Beispiele für die in diesem Abschnitt behandelten Anweisungen im Verzeichnis „Reisen“ in der Datei „REISEN96.XLS“ im Modulblatt „Anfang“.

```
Sub auto_open()
  …Anweisungen
End Sub

Sub auto_close()
  …Anweisungen
End Sub
```

Hinweis Es kann pro Arbeitsmappe nur eine „auto_open“- und eine „auto_close“-Prozedur geben. Wird eine Arbeitsmappe durch VBA-Code geladen, werden die „auto“-Prozeduren nicht automatisch ausgeführt. Sie müssen mit Hilfe der „RunAutoMacros“-Methode aufgerufen werden:

In diesem Beispiel wird die Arbeitsmappe ANALYSE.XLS geöffnet und der "Auto-Open"-Makro ausgeführt.

```
Workbooks.Open "ANALYSE.XLS"
ActiveWorkbook.RunAutoMacros xlAutoOpen
```

In diesem Beispiel wird der Makro "Auto_Close" für die aktive Arbeitsmappe ausgeführt und die Arbeitsmappe geschlossen.

© by Josef Broukal, Vienna, Austria

```
With ActiveWorkbook
  .RunAutoMacros xlAutoClose
  .Close
End With
```

Beim Aktivieren und Verlassen eines Tabellenblattes

EXCEL läßt es zu, daß ein VBA-Programm automatisch gestartet wird, sobald ein neues Blatt in den Vordergrund geholt wird. Die Anweisung dafür lautet:

```
Application.OnSheetActivate = ActiveWorkbook.Name & _
  "!Modul1.Meine_Aktivieren_Sub"
```

In diesem Beispiel wird der OnSheetActivate-Eigenschaft der Makro "Meine_Aktivieren_Sub" in Modul1 der aktiven Arbeitsmappe zugeordnet. Dieser Makro wird jedesmal ausgeführt, wenn der Benutzer ein Tabellenblatt in einer beliebigen Arbeitsmappe aktiviert.

In diesem Beispiel wird der OnSheetActivate-Makro entfernt.

```
Application.OnSheetActivate = ""
```

So allgemein wird die „OnSheetActivate“-Eigenschaft wohl kaum nützlich sein. Sie finden in der Datei „REISEN96.XLS“ ein lebensnäheres. Dort geht es darum, daß in einem Arbeitsblatt Nullen angezeigt werden müssen, in einem zweiten aber nicht angezeigt werden dürfen. Jedesmal, wenn zu einem anderen Blatt gewechselt wird, läuft der „OnSheetActivate“-Makro automatisch ab—er prüft dabei, ob es sich um eines der beiden Blätter aus „REISEN96.XLS“ handelt. Falls nicht, stellt er seine Arbeit gleich wieder ein.

© by Josef Broukal, Vienna, Austria

```
'Sub Nullenanzeige()

' In der Tabelle "Reisedaten" muß die Uhrzeit "00:00" angezeigt werden
' können. Diese Uhrzeit entspricht dem Datum 1.1.1900, 00 Uhr, und
' wird durch die Zahl "0" repräsentiert.
' In diesem Blatt müssen daher Nullen angezeigt werden.

' In der Tabelle "SummenTabelle" stören dagegen die Nullen.
' Daher wird bei der Anzeige dieser Tabelle die Nullen-Anzeige
ausgeschaltet.

Sub Nullenanzeige()

  ' OnSheetActivate ist global (für jede geöffnete Mappe) wirksam.
  ' Daher müssen wir prüfen, ob diese Arbeitsmappe aktiv ist.
  If ActiveWorkbook.Name <> ThisWorkbook.Name Then Exit Sub

  If ActiveSheet.Name = "Reisedaten" Then
    ActiveWindow.DisplayZeros = True
    Call AlleBeschriebenenSpaltenZeigen
  ElseIf ActiveSheet.Name = "SummenTabelle" Then
    ActiveWindow.DisplayZeros = False
    Call AlleBeschriebenenSpaltenZeigen
  End If

End Sub
```

Soll ein VBA-Programm beim Verlassen eines Blattes ausgeführt werden, so müssen Sie der „OnSheetDeactivate“-Eigenschaft den Namen einer VBA-Prozedur angeben, die dann jedesmal, wenn Sie von einem Blatt in ein anderes wechseln, ausgeführt wird.

© by Josef Broukal, Vienna, Austria

Einzelfragen

In diesem Abschnitt des Skriptums wollen wir eine Reihe von einzelnen Fragen behandeln und anhand kurzer Beispielsprogramme beantworten. Gemeinsam ist diesen Fragen, daß sie mit der Ablaufsteuerung von VBA–Programmen zu tun haben.

Feststellen, wo der Cursor steht

Den vollständigen Code für diese Beispiele finden Sie im Verzeichnis „Beispiel“ in der Arbeitsmappe „Cursor“ im Modulblatt „WoStehtDerCursor“.

Angenommen, Sie wollen wissen, wo der Benutzer eines von Ihnen geschriebenen VBA–Programmes den Cursor hingesetzt hat—vielleicht hat das für den weiteren Ablauf Ihres Programmes eine Bedeutung.

Komplette Adresse Die Adresse der Zelle, in der der Cursor steht, erfragen Sie mit Hilfe einer Eigenschaft des Window– und des Application-Objektes, der „ActiveCell“–Eigenschaft:

```
MsgBox ActiveCell.Address (external:=True)
```

Angenommen, der Cursor steht in Zelle A4 von „Tabelle1“ der Arbeitsmappe „Mappe1“. Dann ergibt dieser Befehl die Antwort:

```
[Mappe1]Tabelle1!$A$4
```

Zeilennummer Interessiert Sie nur die Nummer der Zeile, dann müssen Sie schreiben:

```
<Variable> = ActiveCell.Row
```

Spaltennummer Hier lautet die Frage so:

```
<Variable> = ActiveCell.Column
```

NAME DER TABELLE

Wenn Sie wissen wollen, in welcher Tabelle der Cursor steht, so müssen Sie den Wert der Eigenschaft „Parent“ („ÜbergeordnetesObjekt“) der Eigenschaft „ActiveCell“ erfragen:

```
<Variable> = ActiveCell.Parent.Name
```

NAME DER ARBEITSMAPPE

Auch hier hilft die „Parent“–Eigenschaft, diesmal auf die Eigenschaft „ActiveSheet“ des Window- oder Application–Objektes angewendet:

```
MsgBox ActiveSheet.Parent.Name
```

Den Cursor versetzen

Angenommen, Sie wollen den Cursor um eine bestimmte Zahl von Zeilen und/oder Spalten versetzen. Das folgende Codestück zeigt, wie es geht:

© by Josef Broukal, Vienna, Austria

ZEILE

```
Sub CursorVersetzen_Zeile()

MsgBox ActiveCell.Address
ActiveCell.Offset(1, 0).Select
MsgBox ActiveCell.Address

End Sub
```

SPALTE

```
Sub CursorVersetzen_Spalte()

MsgBox ActiveCell.Address
ActiveCell.Offset(0, 1).Select
MsgBox ActiveCell.Address

End Sub
```

DIE „OFFSET“–METHODE

Gibt einen Bereich zurück, der sich aus einem Abstand zu einem angegebenen Bereich berechnet.

Syntax

Objekt.Offset(rowOffset, columnOffset)

Elemente

Objekt Erforderlich. Steht als Platzhalter des Bereichs, auf den die Offset-Methode angewandt wird.

rowOffset Optional. Steht als Platzhalter der Anzahl der Zeilen (positiv, negativ oder Null), die den Offset für den Bereich angeben. Ohne Angabe des Arguments wird Null angenommen.

columnOffset Optional. Steht als Platzhalter der Anzahl der Spalten (positiv, negativ oder Null), die den Offset für den Bereich angeben. Ohne Angabe des Arguments wird Null angenommen.

Feststellen, ob umliegende Zellen beschrieben worden sind

Den kompletten Code dieses Beispiels finden Sie im Verzeichnis „Beispiel“ in der Datei „REGION.XLS“.

Oft werden Sie beim Schreiben eines VBA–Programmes nicht wissen, mit welchen Zellen Ihr Programm arbeiten wird müssen. Denken Sie zum Beispiel daran, daß jemand eine Liste (das ist ein geschlossener Bereich von mit Werten versehenen Tabellenzellen) ausfüllt oder aus einer anderen Anwendung übernimmt. Was Sie wahrscheinlich immer wissen, ist, wo die obere linke Zelle eines solchen Bereiches liegt. Von dort ausgehend können Sie VBA die Ausdehnung des Bereiches suchen lassen:

© by Josef Broukal, Vienna, Austria

```
Sub AktuelleRegionWählen()

ThisWorkbook.Activate
Sheets("Tabelle1").Select
[A1].Select
Call BereichAusfüllen
[A1].Select
Selection.CurrentRegion.Select
MsgBox Selection.Address(External:=True)

End Sub



Sub BereichAusfüllen()

Zeilenzahl = CInt(InputBox("Wieviele Zeilen soll der Bereich haben?"))
Spaltenzahl = CInt(InputBox("Und wieviele Spalten?"))

ThisWorkbook.Activate
Sheets("Tabelle1").Select
[A1].Select
For Zeilenzähler = 1 To Zeilenzahl
    For Spaltenzähler = 1 To Spaltenzahl
        ActiveCell.Value = Zeilenzähler * Spaltenzähler
        ActiveCell.Offset(0, 1).Select
    Next
    ActiveCell.Offset(1, -Spaltenzahl).Select
Next

End Sub
```

```
ThisWorkbook.Activate
```

Macht die Arbeitsmappe, in der das VBA–Programm läuft, zur aktiven Arbeitsmappe.. Vergleiche: „ActiveWorkbook“—jede beliebige Arbeitsmappe, die gerade aktiv ist.

```
Sheets("Tabelle1").Select
```

Wählt in der gerade aktiven Arbeitsmappe „Tabelle1“ aus.

```
[A1].Select
```

Wählt in der gerade ausgewählten Tabelle die Zelle A1 aus.

```
Call BereichAusfüllen
```

Ruft die Prozedur „BereichAusfüllen“ aus. Diese wird weiter unten besprochen.

```
[A1].Select
```

Wählt in der gerade ausgewählten Tabelle wieder die Zelle A1 aus.

```
Select.CurrentRegion.Select
```

Wählt alle Zellen in der Nachbarschaft von A1 aus, die nicht leer sind. Siehe Erklärung gleich unten.

© by Josef Broukal, Vienna, Austria

```
MsgBox Selection.Address(external:=True)
```

Anzeige der Adresse der ausgewählten Zellen in der Form Mappe–Tabelle–Zellkoordinaten.

DIE „CURRENTREGION"–EIGENSCHAFT

Gibt ein Bereich-Objekt zurück, das den aktuellen Bereich repräsentiert. Der aktuelle Bereich entspricht der Fläche um die aktuelle Auswahl, die durch eine Kombination aus leeren Zeilen und leeren Spalten beschränkt ist. Nur-Lese-Zugriff.

Syntax

Objekt.CurrentRegion

Elemente

Objekt Erforderlich. Steht als Platzhalter des markierten Bereichs innerhalb des aktuellen Bereichs.

Anmerkungen Diese Eigenschaft bietet sich bei vielen Operationen an, die die Markierung automatisch auf den umgebenden aktuellen Bereich erweitern (zum Beispiel die „AutoFormat"-Methode).

DIE ADDRESS–METHODE

Gibt den Bezug auf einen Bereich als Zeichenfolge in der Sprache des Makros zurück.

Syntax

Objekt.Address(rowAbsolute, columnAbsolute, referenceStyle, external, relativeTo)

Elemente

Die Address-Methode hat folgenden Objektkennzeichner mit nachfolgenden benannten Argumenten:

Objekt Erforderlich. Gibt einen Bezug auf diesen Bereich zurück.

rowAbsolute Optional. Ist das Argument Wahr oder nicht angegeben, wird der Bezugsteil mit der Zeilenangabe als absoluter Bezug zurückgegeben.

columnAbsolute Optional. Ist das Argument Wahr oder nicht angegeben, wird der Bezugsteil mit der Spaltenangabe als absoluter Bezug zurückgegeben.

referenceStyle Optional. Wird xlA1 angegeben oder erfolgt keine Angabe, gibt die Methode einen Bezug im A1-Bezugssystem zurück. Wird xlR1C1 angegeben, so gibt die Methode einen Bezug im Z1S1-Bezugssystem zurück.

External Optional. Ist das Argument Wahr, gibt die Methode einen externen Bezug zurück. Ist das Argument Falsch, gibt die Methode einen lokalen Bezug zurück. Die Voreinstellung ist Falsch.

relativeTo Optional. Sind **rowAbsolute** und **columnAbsolute** beide Falsch und wird für Bezugsart der Wert xlR1C1 angegeben, so müssen Sie einen Startpunkt für den relativen Bezug angeben. Dieses Argument ist ein Bereich-Objekt und definiert den Startpunkt.

© by Josef Broukal, Vienna, Austria

Feststellen, welche Zellen markiert worden sind

Den kompletten Code der Beispiele in diesem Abschnitt enthält die Datei „AUSWAHL.XLS“ im Verzeichnis „Beispiel“.

Angenommen, Ihr Benutzer soll mit der Maus eine Gruppe von Zellen markieren, und dann soll ihr VBA–Programm damit arbeiten. Dann müssen Sie zunächst feststellen, welche Zellen markiert worden sind. Dafür bietet sich die „Auswahl“–Eigenschaft an.

Ein Bereich

Markieren Sie zunächst mit Maus oder Tastatur einige Zellen in Tabelle1 der Datei „AUSWAHL.XLS“ im Verzeichnis „Beispiel“.

Drücken Sie dann den Schaltknopf „EinBereich“.

Das Programm zeigt Ihnen in einem Dialog die Koordinaten der ausgewählten Zellen.

Hier ist der Code dieses Beispiel–Programmes:

```
Sub AuswahlFeststellen_EinBereich()

Set EinBereich = ActiveWindow.Selection
MsgBox EinBereich.Address

End Sub
```

Mehrere Bereiche

Unser kleines VBA–Programm funktioniert nicht, wenn Sie mehrere Bereiche markiert haben. Die „Address“–Methode versagt in diesem Fall. Wir müssen jede der markierten Flächen („Areas“) der Reihe nach betrachten:

```
Sub AuswahlFeststellen_MehrereBereiche()

For Each EinBereich In ActiveWindow.Selection.Areas
    MsgBox EinBereich.Address(External:=True)
Next

End Sub
```

Jede Auswahl hat die Eigenschaft „Areas“ (deutsch „Flächeliste“); diese gibt eine Zahl zurück, die Zahl der unabhängigen Teile einer Auswahl mehrerer Zellbereiche auf einmal. In den Anweisungen

```
Sub AuswahlFeststellen_MehrereBereiche()

For Each EinBereich In ActiveWindow.Selection.Areas
    MsgBox EinBereich.Address(External:=True)
Next

End Sub
```

arbeiten wir Fläche nach Fläche aus der Flächeliste durch. Der Objektvariablen „EinBereich“ wird bei jedem Durchlauf eine Fläche als Objekt zugewiesen. Mithilfe der „Address“–Methode werden die Koordinaten jeder Fläche ausgegeben.

© by Josef Broukal, Vienna, Austria

DIE SELECTION–EIGENSCHAFT

Gibt das gegenwärtig ausgewählte Objekt im aktiven Fenster des Objekts „Application“ („Anwendung“) oder die aktuelle Auswahl des angegebenen Objekts „Window“ („Fenster“) zurück. Nur-Lese-Zugriff.

Syntax

Objekt.Selection

Elemente

Objekt Optional für das Objekt „Application“. Erforderlich für das Objekt „Window“. Steht als Platzhalter des Objekts, das die Auswahl enthält.

Anmerkungen

Der von der Selection-Eigenschaft zurückgegebene Objekttyp hängt von der Art der Auswahl ab (wenn z.B. in einem Dialogfeld ein Textfeld ausgewählt ist, gibt diese Eigenschaft ein Textfeld-Objekt zurück). Die Selection-Eigenschaft gibt „Nothing“ zurück, wenn nichts ausgewählt wurde.

DIE AREAS–EIGENSCHAFT

Gibt einen einzelnen Bereich (ein „Range“-Objekt, Syntax 1)oder eine Auflistung aller Bereiche (ein „Areas“-Objekt, Syntax 2) in einer Mehrfachauswahl zurück.

Syntax 1

Objekt.Areas(Index)

Syntax 2

Objekt.Areas

Elemente

Objekt Erforderlich. Steht als Platzhalter des Bereichs der Mehrfachauswahl.

Index Erforderlich für Syntax 1. Steht als Platzhalter der Nummer des Bereichs innerhalb der Mehrfachauswahl.

Anmerkungen

Bei einer einfachen Auswahl gibt die Areas-Methode eine Auflistung eines Objekts (das ursprüngliche Bereichs-Objekt) zurück. Bei einer Mehrfachauswahl gibt die Areas-Methode eine Auflistung zurück, die jeweils ein Objekt für jede Auswahl enthält.

Auf Arbeitsblättern navigieren

Zur letzten benützten Zelle springen Dazu wenden wir die „SpecialCells“–Methode an. Achtung: Diese Methode kann nur auf Bereichs–Objekte angewendet werden. Wir schreiben daher:

© by Josef Broukal, Vienna, Austria

```
Sub LetzteZelle()
  ActiveSheet.Cells.SpecialCells(xlLastCell).Select
End Sub
```

Hinweis Eine Zelle ist für EXCEL auch dann „benützt“, wenn Ihr Inhalt gelöscht worden ist. Erst nach Speichern und Wiederaufrufen der Arbeitsmappe wird die (von links oben gesehen) letzte mit Inhalt belegte Zelle die „xlLastCell“.

Hinweis Die „SpecialCells“-Methode funktioniert nicht einwandfrei. Sie schießt gelegentlich übers Ziel hinaus und liefert eine Adresse unterhalb bzw. rechts der tatsächlich letzten belegten Zelle in einer Tabelle. Im Internet ist folgender Code veröffentlicht worden, der diesen Fehler in EXCEL 5.0 und EXCEL 7.0 umgeht:

```
Function RealLastCell(TheSheet As Worksheet) As Range
'   Returns a single-cell range object that represents
'   the intersection of the last non-empty row and the
'   last non-empty column

    Dim ExcelLastCell As Range
    Dim Row As Integer, Col As Integer
    Dim LastRowWithData As Integer, LastColWithData As Integer

    Application.ScreenUpdating = False

'   ExcelLastCell is what Excel thinks is the last cell
    Set ExcelLastCell = TheSheet.Cells.SpecialCells(xlLastCell)

'   Determine the last row with data in it
    LastRowWithData = ExcelLastCell.Row
    Row = ExcelLastCell.Row
    Do While Application.CountA(TheSheet.Rows(Row)) = 0 And Row <> 1
        Row = Row - 1
    Loop
    LastRowWithData = Row

'   Determine the last column with data in it
    LastColWithData = ExcelLastCell.Column
    Col = ExcelLastCell.Column
    Do While Application.CountA(TheSheet.Columns(Col)) = 0 And Col <> 1
        Col = Col - 1
    Loop
    LastColWithData = Col

'   Create the range object
    Set RealLastCell = TheSheet.Cells(Row, Col)
End Function
```

Hinweis Die Function „CountA“ ist eine Tabellen-Funktion. Sie heißt im deutschen EXCEL „Anzahl2“.

Zur letzten beschriebenen Zelle in einer Spalte springen Dazu müssen wir ein wenig hin- und herspringen: Zunächst in die letzte Zeile; ist diese Zelle in der Spalte leer, dann wieder nach oben:

© by Josef Broukal, Vienna, Austria

```
Sub LetzteZelleInSpalte()
  ActiveSheet.Cells(Rows.Count, _
    ActiveCell.Column).Select
  If ISBLANK(ActiveCell) Then
    ActiveCell.End(xlUp).Select
  End If
End Sub
```

Hinweis Wenn eine ganze Zeile markiert ist, bevor „End(xlup)“ angeordnet wird, verkürzt EXCEL die Auswahl auf die erste Spalte der Zeile und sucht dann erst nach der von unten her gesehenen ersten beschriebenen Zelle:

```
Sub try()

Set Testzeile = ActiveSheet.Rows(13)      'Zeile enthält nur leere Zellen
Testzeile.Select
MsgBox Testzeile.Address
Set Testzeile = Testzeile.End(xlUp)       'Letzte beschriebene Zeile suchen
MsgBox Testzeile.Address                  'Ausgewählt ist nur noch Spalte A!
Testzeile.Select

End Sub
```

Zur letzten beschriebenen Zelle in einer Zeile springen Analog zum gerade beschriebenen Beispiel lautet der Code:

```
Sub LetzteZelleInZeile()
  ActiveSheet.Cells(ActiveCell.Row, _
    Columns.Count).Select
  If ISBLANK(ActiveCell) Then
    ActiveCell.End(xlToLeft).Select
  End If
End Sub
```

Den gesamten benützten Zellbereich feststellen Dafür bietet VBA für EXCEL die Eigenschaft „UsedRange“ an:

```
Sub VerwendeterBereichFeststellen()
  MsgBox Sheets("Tabelle1").UsedRange.Address
End Sub
```

Hinweis Eine Zelle gilt auch dann als verwendet, wenn ihr Inhalt seit dem letzten Laden gelöscht worden ist. Sollen gelöschte Zellen nicht mehr „verwendet“ worden sein, müssen Sie die Arbeitsmappe speichern, schließen und wieder öffnen.

Ausschneiden, Kopieren und Einfügen

Beispiels–Code zu diesem Abschnitt finden Sie im Verzeichnis „Beispiel“ in der Datei „KOPIER.XLS“.

Sie kennen das sicher vom interaktiven Betreib von EXCEL: Sobald Sie anordnen, daß eine Zelle ausgeschnitten oder kopiert werden soll, erscheint um sie ein Laufrahmen. Selbst wenn Sie „Ausschneiden“ gewählt haben, bleibt der Zellinhalt zunächst erhalten. Erst wenn Sie eine andere Zelle zur aktiven Zelle machen und dann „Einfügen“ anordnen, wird tatsächlich ausgeschnitten oder kopiert.

© by Josef Broukal, Vienna, Austria

Bei „Ausschneiden“ verschwindet der Laufrahmen, sobald Sie das Ausgeschnitte wieder eingefügt haben. Bei „Kopieren“ bleibt der Laufrahmen auch nach dem Einfügen erhalten. Sie können ihn nur durch Drücken der ESC–Taste oder Wechseln zu einer anderen Aktion beseitigen.

Solange Sie den Laufrahmen sehen, befindet sich EXCEL im „AusschneidenKopieren–Modus“. Das gleich gilt für Ausschneiden und Kopieren mittels VBA–Code.

Den Inhalt von Zellen ausschneiden—„Cut“

“Ausschneiden“ führt dazu, daß der Inhalt von Zellen in die Zwischenablage gestellt wird. Erst durch Einfügen aus der Zwischenablage wird der Zellinhalt wirklich versetzt. Die „Ausschneiden“–Methode hat folgende Syntax:

`Objekt.Cut`*`(destination)`*

Angenommen, Sie wollen den Inhalt von Zelle E7 im aktiven Blatt ausschneiden. Dann müssen Sie schreiben:

```
Sub Cut()
  ActiveSheet.Cells(7, 5).Cut
End Sub
```

Der Zellinhalt bleibt erhalten, lediglich der Laufrahmen um die Zelle herum zeigt an, daß ihr Inhalt in die Zwischenablage kopiert wurde.

Um den Zellinhalt in einem Arbeitsgang in Zelle A1 wieder einzufügen, müssen Sie schreiben:

```
Sub AusschneidenUndEinfügen()
  ActiveSheet.Cells(7, 5).Cut Destination:=Range("A1")
End Sub
```

Hinweis Der Inhalt der Zelle(n) im Zielbereich wird überschrieben.

Hinweis Die Zwischenablage ist nach dem Einfügen im Zielbereich wieder leer.

Den Inhalt von Zellen kopieren—„Copy“

Im Unterschied zum „Ausschneiden“ bleibt nach dem Einfügen der Inhalt der kopierten Zelle erhalten. Kopieren und Einfügen wird in einem so angeordnet:

```
Sub KopierenUndEinfügen()
  ActiveSheet.Cells(7, 5).Copy Destination:=Range("A1")
End Sub
```

Wollen Sie den Zielbereich nicht gleich angeben, müssen Sie das benannte Argument „Destination“ (Zielbereich) weglassen.

Hinweis Der Inhalt der Zelle(n) im Zielbereich wird überschrieben.

Hinweis Die Zwischenablage ist nach dem Einfügen im Zielbereich wieder leer. Darin unterscheidet sich der VBA-Befehl vom interaktiven Kopieren, wo mehrfach eingefügt werden kann.

© by Josef Broukal, Vienna, Austria

Den Inhalt der Zwischenablage einfügen—„Paste“

Die dafür zu verwendende Methode heißt „Paste“. Sie entnimmt den Inhalt der Zwischenablage und fügt ihn in ein Arbeitsblatt ein.

```
Worksheets("Tabelle1").Range("C1:C5").Copy
ActiveSheet.Paste destination:=Worksheets("Tabelle1").Range("D1:D5")
```

Bevor Sie die „Paste“–Methode anwenden, muß ein Zellinhalt mit „Cut“ oder „Copy“ in die Zwischenablage gestellt worden sein. (Sie können beide Methoden verwenden, ohne einen Zielbereich anzugeben.)

Ist die Zwischenablage leer, fügt EXCEL leere Zellen ein. Die „Paste“–Methode kann also auch dazu verwendet werden, in Tabellenblättern leeren Platz zu schaffen.

Für das Einzufügende Platz machen—„Insert“

Die entweder als „destination“ bei „Cut“ und „Copy“ angegebenen oder mit „Paste“ einfügten Inhalte überschreiben die vorhandenen Zellinhalte. Wenn Sie wollen, daß für das Einzufügende Platz geschaffen wird, müssen Sie statt „Paste“ die Methode „Insert“ verwenden.

Die „Insert“–Methode verschiebt die Zelle, an deren Stelle aus der Zwischenablage eingefügt wird, nach rechts oder nach unten. EXCEL entscheidet das entweder automatisch, oder Sie geben es mit den benannten Argumenten „xlToRight“ bzw. „xlDown“ vor.

In diesem Beispiel wird eine neue Zeile unterhalb von Zeile 3 eingefügt.

```
Rows(4).Insert
```

In diesem Beispiel wird im Bereich A1:C5 eine Zelle eingefügt. Die übrigen Zellen werden nach unten verschoben.

```
Range("A1:C5").Insert (xlDown)
```

In diesem Beispiel wird eine ganze Zeile an der derzeit markierten Position eingefügt. (Das macht wahrscheinlich nur Sinn, wenn die Zwischenablage leer ist.)

```
Selection.EntireRow.Insert
```

Hinweis Die in die Zwischenablage gestellten Daten etc. werden beim Einsetzen in einen Zielbereich aus der Zwischenablage wieder entnommen. Es kann also alles, was ausgeschnitten bzw. kopiert wurde, nur einmal eingefügt werden.

Mehr zu diesem Thema in der Visual-Basic-Online-Hilfe unter „Methods“, „Cut–Methode“, „Copy–Methode“, „Paste–Methode“, „Insert-Methode“ und „CutCopyMode–Eigenschaft“

Ein Bereichs–Objekt erzeugen

Den kompletten Code der Beispiele in diesem Abschnitt finden Sie im Verzeichnis „Beispiel“ in der Datei „OBJEKT.XLS“.

Mit Visual Basic steuern oder schaffen Sie EXCEL–Objekte, benützen deren Methoden und lesen oder setzen deren Eigenschaften. Sie können aber auch selbst Objekte erzeugen.

© by Josef Broukal, Vienna, Austria

Angenommen, Sie wollen eine Zelle in der aktiven Tabelle auswählen. Das geht recht einfach:

```
[A1].Select
```

Liegt diese Zelle aber nicht in der aktiven Arbeitsmappe, wird die Ansprache mühsamer:

```
Workbooks("OBJEKT.XLS").Activate
Worksheets("Tabelle1").Select
Cells(1, 1).Select
```

Stellen Sie sich vor, Sie müssen zwanzig Mal eine Zelle in einer nicht aktiven Arbeitsmappe ansprechen! Was für eine Schreibarbeit! Und wie schlecht lesbar! Außerdem führt dieser Code dazu, daß Arbeitsmappe und Tabelle in den Vordergrund geholt werden—das kann passen, muß es aber nicht.

Zum Glück bietet uns VBA eine Lösung an. Wir können ein Objekt einer Variablen zuweisen und es ab diesem Zeitpunkt durch Angabe des Variablennamens manipulieren.

DIE „SET“–ANWEISUNG

Wenn Zellen ausgewählt worden sind, lohnt es sich häufig, sie als Bereichs–Objekt anzusprechen und mit den Methoden und Eigenschaften des Bereichs–Objektes zu arbeiten. Für das Schaffen von Objekten ist die „Set“–Anweisung gedacht:

Set Variable=Objekt

```
Sub Objekt_1()

Set MeinBereich = Workbooks("Objekt.XLS"). _
    Sheets("Tabelle1").Cells(1, 1)
MeinBereich.Value = "Diese Meldung kommt aus Zelle A1 von Tabelle1, " & _
    "Arbeitsmappe Objekt.XLS"
MsgBox MeinBereich.Value

End Sub
```

Enthält der Bereich, aus dem ein Objekt gebildet werden soll, mehrere Zellen, so muß—im Gegensatz zum VBA–Handuch für EXCEL 5.0, Seite 116, das Objekt so gebildet werden:

```
Set MeinObjekt = Workbooks("OBJEKT.XLS").Worksheets("Tabelle1"). _
    Range(Worksheets("Tabelle1").Cells(1; 1); _
    Worksheets("Tabelle1"). Cells(8; 1))
```

Mit Hilfe der „Set“–Anweisung können wir Kopien von Objekten erzeugen:

```
Set DeinObjekt = ActiveWindow.Selection
Set TestObjekt = DeinObjekt  ' Objektreferenz zuweisen.
' TestObjekt und DeinObjekt beziehen sich auf dasselbe Objekt.
DeinObjekt.Text = "Hallo Welt"  ' Eigenschaft initialisieren.
TestZnF = TestObjekt.Text  ' "Hallo Welt" zurückgeben
' Zuordnung beenden. TestObjekt bezieht sich nicht mehr auf DeinObjekt.
Set TestObjekt = Nothing
```

Hinweis Verwenden wir bei Bereichs–Objekten die Anweisung

Variable=Objekt

© by Josef Broukal, Vienna, Austria

dann erhält „Variable“ den Wert des Objektes. Es wird also keine Objektvariable geschaffen!

NUR EINEN TEIL EINES OBJEKTES ZUWEISEN

In unserem Beispiel oben:

```
Set MeinBereich = Workbooks("Objekt.XLS"). _
    Sheets("Tabelle1").Cells(1, 1)
```

enthält „MeinBereich“ die feste Zuordnung an Zelle A1 von „OBJEKT.XLS“, „Tabelle1“. Was ist, wenn wir aber verschiedene Zellen in diesem Tabellenblatt ansprechen wollen? Diesfalls wäre es klüger, der Objektvariablen nur die Referenz auf Mappe und Blatt zuzuweisen:

```
Sub Objekt_2()

Set Meinblatt = Workbooks("Objekt.XLS"). _
    Sheets("Tabelle1")
Meinblatt.Cells(1, 1).Value = " Diese Meldung kommt aus Zelle A1 von
Tabelle1, " & _
    "Arbeitsmappe Objekt.XLS"
MsgBox Meinblatt.Cells(1, 1).Value

Meinblatt.Cells(1, 1).Font.Bold = True

End Sub
```

Dies gibt uns die Möglichkeit, zum Beispiel anschließend zu schreiben:

```
MeinBlatt.Cells(1, 1).Font.Bold = True
```

EINE UNABHÄNGIGE KOPIE EINES OBJEKTES ERZEUGEN

```
Wie wir weiter oben gesehen haben, können wir mit der Anweisung
```

Set Objektvariable2=Objektvariable1

die Kopie eines Objektes erzeugen. Weisen wir VBA an:

`Objektvariable2.Value=Wert`

dann erhält auch Objektvariable1 diesen Wert, den sie verweist ja auf die selbe Zelle.

```
Sub Objekt_3()

Set Originalbereich = Workbooks("OBJEKT.XLS"). _
    Sheets("Tabelle1").Cells(1, 1)
Set Kopiebereich = Originalbereich
Originalbereich.Value = "Kopiebereich enthält den Wert von Originalbereich"
MsgBox Kopiebereich.Value

End Sub
```

Ändern wir die Größe von „Originalbereich“, behält „Kopiebereich“ seine Größe:

© by Josef Broukal, Vienna, Austria

```
Sub Objekt_4()

Set Originalbereich = Workbooks("OBJEKT.XLS"). _
    Sheets("Tabelle1").Cells(1, 1)
Set Kopiebereich = Originalbereich
Set Originalbereich = Originalbereich.Offset(3, 3)
MsgBox "Originalbereich hat die Adresse: " & _
    Originalbereich.Address
MsgBox "Kopiebereich hat die Adresse: " & _
    Kopiebereich.Address

End Sub
```

Bewegen wir den Zellzeiger um *x* Zeilen und *y* Spalten vom Objekt weg, dann bleibt das Objekt an seiner alten Stelle:

```
Sub Objekt_5()

Workbooks("OBJEKT.XLS").Activate
Sheets("Tabelle1").Select
Cells(1, 1).Select
Set Originalbereich = Selection
Originalbereich.Offset(3, 3).Select
MsgBox "Originalbereich hat die Adresse: " & _
    Originalbereich.Address
MsgBox "Aber Cursor steht in " & _
    ActiveCell.Address

End Sub
```

Versetzen wir „Originalbereich“, bleibt „Kopiebereich“ am ursprünglichen Ort von „Originalbereich“:

```
Sub Objekt_6()

Workbooks("OBJEKT.XLS").Activate
Sheets("Tabelle1").Select
Cells(1, 1).Select
Set Originalbereich = Selection
Set Kopiebereich = Originalbereich
Set Originalbereich = Selection.Offset(3, 3)
MsgBox "Originalbereich wurde versetzt nach: " & _
    Originalbereich.Address
MsgBox "Kopiebereich hat nach wie vor die " & _
    "Adresse: " & Kopiebereich.Address

End Sub
```

Grösse eines Bereichs–Objektes ändern

Dazu verwenden wir die „Resize“-Methode. Aus der VBA–Hilfe stammt die folgende Erklärung:

RESIZE–METHODE

Ändert die Größe des Bereiches.

Syntax

© by Josef Broukal, Vienna, Austria

Objekt.Resize(*rowSize, columnSize*)

Elemente

Die Resize -Methode hat folgenden Objektkennzeichner mit nachfolgenden benannten Argumenten:

Objekt Erforderlich. Steht als Platzhalter des Bereichs-Objekts („Range“), dessen Größe geändert werden soll.

RowSize Optional. Steht als Platzhalter der Anzahl der Zeilen im neuen Bereich. Falls nicht angegeben, bleibt die Anzahl der Zeilen im Bereich unverändert.

ColumnSize Optional. Steht als Platzhalter der Anzahl der Spalten im neuen Bereich. Falls nicht angegeben, bleibt die Anzahl der Spalten im Bereich unverändert.

Hinweis Fügen wir mitten in die Zellen eines Bereiches eine neue Zelle ein, dann wird das Bereichs–Objekt um diese Zeile größer:

```
Sub Objekt_7()

Set MeinObjekt = Workbooks("OBJEKT.XLS"). _
  Worksheets("Tabelle1"). _
    Range(Worksheets("Tabelle1").Cells(1, 2), _
    Worksheets("Tabelle1").Cells(8, 2))
MsgBox "MeinObjekt umfaßt: " & MeinObjekt.Address(External:=True)
Set Einfügebereich = MeinObjekt.Cells(5, 1)
Einfügebereich.Insert Shift:=xlDown
MsgBox "MeinObjekt umfaßt nach Einfügen einer Zelle: " & _
    MeinObjekt.Address(External:=True)

End Sub
```

Bitte beachten Sie:

Wird in diesem Beispiel eine Zelle in der ersten Zeile des Bereichs–Objekts eingefügt, wird das Objekt um eine Zeile nach unten verschoben.

Erfolgt das Einfügen in der zweiten bis letzten Zeile, wird das Bereichs–Objekt vergrößert.

Sie können das testen, indem Sie in der ersten fettgedruckten Zeile des Codes statt „5“ zunächst „1“, dann „8“ eingeben.

Verschieben eines Bereichs–Objektes

Wollen Sie das Bereichs–Objekt verschieben, müssen Sie es
- auswählen
- die ausgewählten Zellen versetzen
- die versetzten Zellen neuerlich der Objektvariablen zuweisen.

Den zugehörigen Code enthält „Sub Objekt_6()“ weiter oben.

© by Josef Broukal, Vienna, Austria

Eine Meldung auf dem Bildschirm anzeigen

Meldungen an den Benützer bieten sich in vielen Situationen an. Wenn Ihr Programm „ein Problem hat“, machen Sie darauf aufmerksam. Läuft eine längere Verarbeitung, verkürzen Sie die Wartezeit und geben dem Benützer die Sicherheit, daß das Programm arbeitet. Wir unterscheiden:

- Meldungen, die den Programmlauf unterbrechen
- Meldungen, die Ihr Programm weiter arbeiten lassen

„MsgBox“—Programm–unterbrechende Meldungen

VBA stellt uns hier die Funktion „MsgBox“ zur Verfügung, deren Erscheinungsbild Sie aus der interaktiven Arbeit mit EXCEL und anderen Windows-Programmen sicher schon kennen: „MsgBox“ zeigt eine Meldung in einem Dialogfeld an und wartet auf die Auswahl einer Schaltfläche. Die Funktion liefert einen Wert, der die ausgewählte Schaltfläche bestimmt.

Das folgende Beispiel für das Programmieren eines Meldungs–Dialogs finden Sie auf der Diskette im Verzeichnis „Beispiel“ in der Datei „MELDUNG.XLS“

```
Sub MeldungsDlg_Anzeigen()

Mldg = "Wollen Sie fortfahren ?"    ' Meldung definieren.
Stil = vbYesNo + vbCritical + vbDefaultButton1
        ' Schaltflächen definieren.
Title = "MeldungsDlg-Demo"  ' Titel definieren.
Help = "DEMO.HLP"  ' Hilfedatei definieren.
Kontext = 1000  ' Kontext für Thema definieren
        ' Meldung anzeigen.
Result = MsgBox(Mldg, Stil, Title, Help, Kontext)
If Result = vbYes Then   ' Benutzer hat "Ja" gewählt.
    TestZeichenF = "Ja" ' entsprechende Operationen ausführen.
Else   ' Benutzer hat "Nein" gewählt.
    TestZeichenF = "Nein"   ' entsprechende Operationen ausführen.
End If
MsgBox "<<Ergebnis>> Is: " & Result

End Sub
```

Syntax

MsgBox(Prompt[,buttons][,title][,helpfile,context])

Elemente

Die Funktion MsgBox verwendet die folgenden benannten Argumente:

Argument	Beschreibung
Prompt	Zeichenfolgenausdruck, der als Meldung im Dialogfeld erscheint. Die Maximallänge von Meldung ist etwa 1024 Zeichen, abhängig von der Breite der verwendeten Zeichen. Wenn Meldung aus mehreren Zeilen bestehen soll, müssen Sie die Zeilen mit einem Wagenrücklaufzeichen (Zeichencode 13) oder einer Folge aus Wagenrücklaufzeichen und Zeilenvorschubzeichen (Zeichencode 10) voneinander trennen.
Buttons	Numerischer Ausdruck mit einem kombinierten Wert, der die Anzahl und den Typ der Schaltflächen, das verwendete Symbol, die voreingestellte

© by Josef Broukal, Vienna, Austria

Argument	Beschreibung
	Schaltfläche und die Bindung des Dialogfelds angibt. Wenn Sie Schaltflächen nicht angeben, ist die Voreinstellung 0.
Title	Zeichenfolgenausdruck, der in der Titelleiste des Dialogfeldes erscheint. Wenn Sie Title nicht angeben, erscheint kein Text in der Titelleiste.
Helpfile	Zeichenfolgenausdruck, der die Hilfedatei mit der kontextbezogenen Hilfe für das Dialogfeld angibt. Wenn Sie Helpfile angeben, müssen Sie auch Context angeben.
Context	Numerischer Ausdruck mit der Zahl für den Hilfekontext, die der Autor der Hilfe für das entsprechende Hilfethema vergeben hat. Wenn Sie Context angeben, müssen Sie auch Helpfile angeben.

Die Werte für das benannte Argument „Buttons“ sind:

Konstante	Wert	Beschreibung
vbOKOnly	0	Nur "OK" anzeigen.
vbOKCancel	1	"OK" und "Abbrechen" anzeigen.
vbAbortRetryIgnore	2	"Abbrechen", "Wiederholen" und "Ignorieren" anzeigen.
vbYesNoCancel	3	"Ja", "Nein" und "Abbrechen" anzeigen.
vbYesNo	4	"Ja", und "Nein" anzeigen.
vbRetryCancel	5	"Wiederholen" und "Abbrechen" anzeigen.
vbCritical	16	Stop-Symbol anzeigen.
vbQuestion	32	Fragezeichen-Symbol anzeigen.
vbExclamation	48	Ausrufezeichen-Symbol anzeigen.
vbInformation	64	Info-Symbol anzeigen.
vbDefaultButton1	0	Erste Schaltfläche ist Voreinstellung.
vbDefaultButton2	256	Zweite Schaltfläche ist Voreinstellung.
vbDefaultButton3	512	Dritte Schaltfläche ist Voreinstellung.
vbApplicationModal	0	An die Anwendung gebunden. Der Benutzer muß das Meldungsfeld beantworten, bevor er seine Arbeit mit der aktuellen Anwendung fortsetzen kann.
vbSystemModal	4096	An das System gebunden. Alle Anwendungen werden angehalten, bis der Benutzer das Meldungsfeld beantwortet.

Die erste Gruppe von Werten (0-5) beschreibt die Anzahl und den Typ der im Dialogfeld angezeigten Schaltflächen. Die zweite Gruppe (16, 32, 48, 64) beschreibt die Symbolart. Die dritte Gruppe (0, 256, 512) legt die voreingestellte Schaltfläche fest. Die vierte Gruppe (0, 4096) legt fest, in welcher Form das Dialogfeld gebunden ist. Verwenden Sie beim Kombinieren von Zahlen zu einem Gesamtwert für das Argument „Buttons“ nur eine Zahl aus jeder Gruppe.

Hinweis Diese Konstanten sind durch Visual Basic festgelegt. Daher können Sie sie an einer beliebigen Stelle im Code anstelle der tatsächlichen Werte verwenden.

© by Josef Broukal, Vienna, Austria

RÜCKGABEWERTE

Konstante	Wert	gedrückte Schaltfläche
vbOK	1	"OK"
vbCancel	2	Abbrechen
vbAbort	3	Abbruch
vbRetry	4	Wiederholen
vbIgnore	5	Ignorieren
vbYes	6	Ja
vbN0	7	Nein

Anmerkungen

Wenn Sie „Helpfile“ und „Context“ angeben, erscheint die Schaltfläche "Hilfe" automatisch im Dialogfeld.

Wenn im Dialogfeld die Schaltfläche "Abbrechen" erscheint, hat das Drücken der Taste ESC dieselbe Wirkung wie das Klicken auf "Abbrechen". Wenn im Dialogfeld die Schaltfläche "Hilfe" erscheint, wird darüber hinaus eine kontextbezogene Hilfe zur Verfügung gestellt. Ein Wert wird aber nur zurückgegeben, wenn eine der anderen Schaltflächen ausgewählt wird.

Nicht–unterbrechende Meldung

Die Statusleiste—am unteren Rand des EXCEL–Fensters—versorgt Sie während der interaktiven Arbeit laufend mit Informationen über das, was EXCEL gerade tut. Mit Hilfe der Eigenschaft „Statusbar“ können Sie, während eine VBA–Programm läuft, dort Meldungen anzeigen; das Programm läuft weiter.

DIE STATUSBAR-EIGENSCHAFT

Syntax

Application. StatusBar

Beispiel

In diesem Beispiel werden die Zahlen von 1 bis 400 erzeugt und in der Statuszeile angezeigt. Danach wird wieder auf die Anzeige der normalen EXCEL-Statusmeldungen umgeschaltet:

© by Josef Broukal, Vienna, Austria

```
Sub StatusBar()

' Hält fest, ob die Statusleiste angezeigt wird oder nicht
AlterZustandStatusLeiste = Application.DisplayStatusBar
' Statusleiste auf alle Fälle anzeigen
Application.DisplayStatusBar = True
' Selbst-definierte Statusmeldung anzeigen
' Arbeit verrichten, während Anzeige sichtbar ist
For Zähler = 1 To 400
    Application.StatusBar = Zähler
Next
' Selbst-definierte Statusmeldung löschen
Application.StatusBar = False
' Alten Zustand (angezeigt/nicht angezeigt)
' wiederherstellen
Application.DisplayStatusBar = AlterZustandStatusLeiste

End Sub
```

DIE „DISPLAYSTATUSBAR"–EIGENSCHAFT

True, wenn die Statusleiste angezeigt wird, **False**, wenn Sie fehlt. Schreib-/Lese-Zugriff.

Nicht–unterbrechende Meldung in EXCEL–Tabelle

An der Statuszeilen–Meldung werden Sie zurecht kritisieren, daß sie kaum bemerkt wird. Mit ein wenig Arbeit können Sie aber eine deutlich sichtbare Meldung erzeugen. Wir benötigen dazu ein EXCEL–Tabellenblatt und ein wenig Code. Angenommen, die Meldung soll in Zelle B4 erscheinen, dann müssen wir schreiben:

```
Sub MeldungInTabelle()
Set AltePosition = ActiveCell
Sheets("Tabelle1").Select
For Zähler = 1 To 200
    [B4].Value = Zähler
Next
Application.GoTo Reference:=AltePosition
End Sub
```

In der Zeile „Set AltePosition…" halten wir fest, in welcher Tabelle der Cursor gerade wo steht. Dorthin gehen wir in der Zeile „Application.Goto…" zurück.

Eine Eingabe durch den Benützer erhalten

EXCEL bietet uns dafür zwei Möglichkeiten:

- benutzerdefinierte Dialogfelder
- die vordefinierte Funktion „InputBox" und die Methode „InputBox"

Für benutzerdefinierte Dialogfelder ist ein eigenes Kapitel dieses Skriptums vorgesehen. Sie bieten die meisten Möglichkeiten, erfordern aber sehr viel Zeichen- und Programmierarbeit. Einfacher kommen sie mit den vordefinierten Eingabefunktionen aus.

Den kompletten Code der Beispiele in diesem Abschnitt finden Sie im Verzeichnis „Beispiel" in der Datei „EINGABE.XLS".

© by Josef Broukal, Vienna, Austria

DIE FUNKTION „INPUTBOX“

Zeigt eine Eingabeaufforderung in einem Dialogfeld an, wartet auf die Eingabe eines Textes oder die Auswahl einer Schaltfläche und liefert den Inhalt des Bearbeitungsfeldes zurück.

Syntax

Variable = InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])

Elemente

Die Funktion InputBox verwendet die folgenden benannten Argumente:

Argument	Beschreibung
Prompt	Zeichenfolgenausdruck, der als Eingabeaufforderung im Dialogfeld erscheint. Die Maximallänge ist etwa 1024 Zeichen, abhängig von der Breite der verwendeten Zeichen. Wenn Prompt aus mehreren Zeilen besteht, müssen Sie die Zeilen mit einem Wagenrücklaufzeichen (Zeichencode 13) oder einer Folge aus Wagenrücklaufzeichen und Zeilenvorschubzeichen (Zeichencode 10) trennen.
Title	Zeichenfolgenausdruck, der in der Titelleiste des Dialogfeldes erscheint. Wenn Sie keinen Titel angeben, erscheint kein Text in der Titelleiste.
Default	Zeichenfolgenausdruck, der als Voreinstellung im Bearbeitungsfeld angezeigt wird, wenn keine Eingaben stattfinden. Wenn Sie Default nicht angeben, erscheint das Textfeld ohne Text.
XPos	Numerischer Ausdruck, der in Twips den horizontalen Abstand vom linken Rand des Dialogfeldes zum linken Rand des Bildschirms festlegt. Wenn Sie XPos nicht angeben, erscheint das Dialogfeld horizontal zentriert.
YPos	Numerischer Ausdruck, der in Twips den vertikalen Abstand vom oberen Rand des Dialogfeldes zum oberen Rand des Bildschirms festlegt. Wenn Sie YPos nicht angeben, erscheint das Dialogfeld etwa ein Drittel unterhalb des oberen Bildschirmrands (bezogen auf die gesamte Bildschirmhöhe).
Helpfile	Zeichenfolgenausdruck, der die Hilfedatei mit der kontextbezogenen Hilfe für das Dialogfeld angibt. Wenn Sie Helpfile angeben, müssen Sie auch Context angeben.
Context	Numerischer Ausdruck mit der Zahl für den Hilfekontext, die der Autor der Hilfe für das entsprechende Hilfethema vergeben hat. Wenn Sie Context angeben, müssen Sie auch Helpfile angeben.

Rückgabe

„InputBox“ gibt immer eine Zeichenfolge zurück. Brechen Sie die Eingabe ab, wird eine leere Zeichenfolge zurückgegeben. Dies gilt auch für den Fall, daß Sie einen Vorgabewert („Default“) an die Funktion übergeben haben.

Anmerkungen

Wenn Sie Helpfile und Context angeben, erscheint die Schaltfläche "Hilfe" automatisch im Dialogfeld. Wenn der Benutzer "OK" wählt oder die EINGABETASTE drückt, liefert die Funktion InputBox den Inhalt des Textfeldes. Wählt der Benutzer "Abbrechen", so liefert die Funktion eine leere Zeichenfolge ("").

Beispiel

© by Josef Broukal, Vienna, Austria

```
Sub Eingabe_Test()

Dim Vorgabetext
Dim Eingabetext

Vorgabetext = True

Eingabetext = InputBox("Wieviel Kopien wollen Sie drucken?", _
    "Dienstplan für dieses Monat drucken", _
    Vorgabetext)

If Eingabetext = "" Then Eingabetext = "Leere Zeichenfolge"

MsgBox "Zurückgegeben wurde: " & TypeName(Eingabetext) & _
    " mit dem Inhalt: " & Eingabetext

End Sub
```

Mehr zu diesem Thema in der Visual-Basic-Online-Hilfe unter „Funktionen“, „InputBox“

DIE METHODE „INPUTBOX“

unterscheidet sich von der gleichnamigen Funktion vor allem dadurch, daß Sie festlegen können, daß ein anderer Datentyp als „Zeichenfolge“ verarbeitet werden soll. Sie können also in Grenzen die Plausibilität einer Eingabe durch VBA überprüfen lassen.

Syntax

Objekt.InputBox(prompt, title, default, left, top, helpFile, helpContextID, type)

Elemente

Die Methode „InputBox" hat folgenden Objektkennzeichner mit nachfolgenden benannten Argumenten:

Objekt Erforderlich. Steht als Platzhalter des Objekts Anwendung.

Prompt Erforderlich. Steht als Platzhalter der im Dialogfeld anzuzeigenden Meldung. Dies kann eine Zeichenfolge, eine Zahl, ein Datum oder ein boolescher Wert sein.

Title Optional. Steht als Platzhalter für die Überschrift des Eingabefelds. Ohne Angabe dieses Arguments erscheint die Standardvorgabe "Eingabe".

Default Optional. Legt den Wert fest, der im Bearbeitungsfeld vorgegeben werden soll, wenn das Dialogfeld auf dem Bildschirm erscheint. Ohne Angabe dieses Arguments bleibt das Bearbeitungsfeld leer. Dieser Wert kann ein Bereich-Objekt sein.

Left Optional. Legt die x-Position des Dialogfelds relativ zur linken oberen Ecke des Bildschirms in Punkten fest. Ein Punkt entspricht 1/72 Zoll (0,35mm).

Top Optional. Legt die y-Position des Dialogfelds relativ zur linken oberen Ecke des Bildschirms in Punkten fest.

© by Josef Broukal, Vienna, Austria

Helpfile Optional. Steht als Platzhalter des Namens der Online-Hilfedatei für dieses Eingabefeld. Werden die Argumente Helpfile und helpContextID angegeben, so erscheint die Hilfe-Schaltfläche im Dialogfeld.

helpContextID Optional. Steht als Platzhalter der Kontextkennung des Hilfethemas in der Hilfedatei, die mit Helpfile angegeben wurde.

Type Optional. Steht als Platzhalter für die Art der zurückgegebenen Daten. Ohne Angabe dieses Arguments gibt das Dialogfeld die Daten in Form von Text zurück. Die folgenden Werte sind für das Argument zulässig:

Wert	Bedeutung
0	Eine Formel
1	Eine Zahl
2	Text (eine Zeichenfolge)
4	Ein Wahrheitswert (Wahr oder Falsch)
8	Ein Zellbezug als ein Bereich-Objekt
16	Ein Fehlerwert, wie z.B. #NV
64	Eine Matrix von Werten

Es kann auch die Summe von zulässigen Werten für das Argument Type benutzt werden. Sollen beispielsweise für ein Eingabefeld Text oder Zahlen zulässig sein, so setzen Sie das Argument Typ3 gleich 1 + 2=3.

Anmerkungen

Mit dem Befehl „InputBox“ können Sie ein einfaches Dialogfeld anzeigen, in das Informationen für ein Programm eingegeben werden können. Das Dialogfeld enthält die Schaltflächen "OK" und "Abbrechen". Wenn "OK" gewählt wird, gibt InputBox den im Dialogfeld eingegebenen Wert zurück. Wird "Abbrechen" gewählt, so gibt InputBox den Wert Falsch zurück.Wenn Type = 0 ist, gibt InputBox die Formel in Textform, z.B. =2*Pl()/360 zurück. Falls die Formel Bezüge enthält, werden diese als A1-Bezugsarten zurückgegeben. (Mit „ConvertFormula“ können Sie zwischen den verschiedenen Bezugsarten wechseln.)Wenn Type = 8 ist, gibt InputBox einen Bereich zurück. Die Set-Anweisung muß verwendet werden, um das Ergebnis einem Bereich zuzuordnen, wie in dem folgenden Beispiel dargestellt.

```
Set MeinBereich = Application.InputBox(Prompt:= _
  "Geben Sie einen Bereich ein:", Type := 8)
```

Wenn Sie die Set-Anweisung nicht verwenden, wird der Variablen der Wert des Bereichs anstelle des Bereichs selbst zugewiesen.

Brechen Sie bei Type:=8 die Eingabe durch Klicken der Schaltfläche „Abbrechen“ ab, entsteht ein Fehler: Erwartet wird eine Zellreferenz, InputBox liefert aber den logischen Wert „False“. Ergebnis: das Programm stoppt und gibt eine Fehlermeldung aus. Die derzeit einzige Abhilfe:

- Fehlerprüfung mit „On Error Resume Next“ ausschalten
- InputBox aufrufen
- Fehlerprüfung mit „On Error Goto 0“ wieder einschalten.
- Prüfen, ob „rGezeigterBereich“ eine Zellreferenz zugewiesen wurde. In diesem Fall wäre sein Variablentyp „Range“; wurde nichts zugewiesen, ist er „Nothing“.

© by Josef Broukal, Vienna, Austria

```
Sub try()

Dim rGezeigterBereich As Range

On Error Resume Next
Set rGezeigterBereich = Application.InputBox _
  (prompt:="Zeigen Sie bitte eine Zelle", Type:=8)
On Error GoTo 0
If Not TypeName(rGezeigterBereich) = "Range" Then
  MsgBox "Keine gültige Eingabe !", vbCritical, "Fehler:"
End If

End Sub
```

Mehr zu diesem Thema in der Visual-Basic-Online-Hilfe unter „Methoden“, „InputBox-Methode“

Dateien kontrolliert öffnen lassen

Müssen während des Programm-Laufes Arbeitsmappen geöffnet werden, dann gibt es dazu zwei Wege:

- Sie können das Dialogfeld anzeigen lassen, das Sie auch im interaktiven Betrieb über das Menü „Datei“ und den Befehl „Öffnen...“ erreichen; in diesem Fall hat der Benützer des Programmes alle Freiheiten.
- Sie können stattdessen die „GetOpenFilename“-Methode verwenden. Sie bietet mehrere Vorteile:
    - Sie öffnet die Datei nicht, sondern stellt lediglich ihren Namen und ihren Pfad zur Verfügung.
    - Sie können im VBA-Code festlegen, welche Art(en) von Dateien gezeigt werden soll(en).
    - Das VBA-Programm kann nach dem Beenden von „GetOpenFilename“ nochmals prüfen, ob die Datei überhaupt geöffnet werden darf.

Den Code zu diesem Abschnitt finden Sie in der Datei „Öffnen.XLS“ im Verzeichnis „Beispiel“

DAS EINGEBAUTE DIALOGFELD „ÖFFNEN“ IST SCHWER ZU KONTROLLIEREN

Das deshalb, weil die Datei beim Beenden des Dialogs gleich geöffnet wird. War es die falsche, muß sie durch VBA-Code wieder geschlossen werden. Wenig elegant, dafür in ein paar Zeilen programmiert:

```
' Bringt eingebauten Dialog "Öffnen" zur Anzeige
' Achtung: Datei wird sofort geöffnet, wenn Benutzer
' die Schaltfläche "Öffnen" anklickt!!!!

Sub DateiOeffnen1()

Dim bDialogErgebnis As Boolean
bDialogErgebnis = Application.Dialogs(xlDialogOpen).Show

' bDialogErgebnis hat Wert True, falls Datei geöffnet wurde,
' False, falls der Dialog abgebrochen wurde

End Sub
```

© by Josef Broukal, Vienna, Austria

DIE METHODE „GETOPENFILENAME“

Eleganter und mit mehr Kontrolle über den „Öffnen“-Vorgang. Ihre Syntax lautet:

Application.GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)

Abbildung 34: „GetOpenFilename“ zeigt zwar den gewohnten Dialog aus „Datei öffnen...“, öffnet aber Dateien nicht, sondern stellt nur ihren Namen zur Verfügung.

„GetOpenFilename“ gibt zurück:

- Schaltfläche „Öffnen“ gedrückt wurde: Laufwerk, Pfad und Name der gewählten Datei. Falls mehrere Dateien ausgewählt wurden (siehe Argument „MultiSelect“ ein paar Absätze weiter unten), ein Datenfeld-jedes Element enthält einen Dateinamen einschließlich Laufwerk und Pfad.
- Schaltfläche „Abbrechen“ gedrückt: den Wahrheitswert False. Falls die auffangende Variable vom Typ „String“ ist, wandelt EXCEL False in den Text „Falsch“ um.

FileFilter Optional. Ein Text, der die Anzeige von Dateien einschränkt. „Filefilter“ hat eine etwas komplizierte Form:

FilterName1, FilterDefinition1, FilterName2, FilterDefinition2, FilterNameN, FilterDefinitonN

Angenommen, Sie wollen nur EXCEL-Arbeitsmappen anzeigen lassen, dann müssen sie als „Filefilter“ angeben:

EXCEL-Arbeitsmappen (*.xls), *.xls

Der Text „EXCEL-Arbeitsmappen (*.xls)“ wird im Feld „Dateityp“ des „GetOpenFilename“-Dialogs angezeigt, die Anzeige von Dateien ist auf Dateien mit der Endung „.xls“ beschränkt.

Hinweise Sie müssen die Datei-Endung, wie oben gezeigt, zweimal angeben. Lassen Sie hinter „EXCEL-Arbeitsmappen“ den Text „(*.xls)“ weg, werden alle Dateien gezeigt!

© by Josef Broukal, Vienna, Austria

Leider ist es nicht möglich, die Anzeige von Dateien durch genauere Angabe eines Dateinamens (etwa „Buch????.xls“) einzuschränken. Soll eine bestimmte Datei-Auswahl erzwungen werden, so muß dies nach dem Schliessen des „GetOpenFilename“-Dialogs geschehen—durch Abprüfen des Namens und Pfades der ausgewählten Datei(en).

Geben Sie „FileFilter“ nicht an, werden alle auch im Datei-Öffnen-Dialog vorgegebenen Dateiarten angezeigt.

FilterIndex Optional. Eine Zahl zwischen 1 und der Anzahl der angegebenen „FileFilter“. Zeigt im Feld „Dateityp“ des „GetOpenFilename“-Dialogs jenen Filternamen an, dessen Reihenfolge in „FileFilter“ der Zahl von „FilterIndex“ entspricht. Wenn nicht angegeben, wird der erste „FileFilter“ gezeigt.

Title Optional. Der Dialogfeld-Titel, z.B.: „Bitte wählen Sie eine Buchhaltungs-Datei aus“

Buttontext Optional. Nur in EXCEL für Apple-Computer verfügbar.

MultiSelect Wenn True, können mehrere Dateien zugleich ausgewählt werden, allerdings nur aus einem Verzeichnis. „GetOpenFilename“ gibt in diesem Fall Pfad und Namen nicht als Text zurück, sondern als Datenfeld—auch wenn nur eine Datei ausgewählt wurde.

Die notwendigen Arbeitsschritte

1. Halten Sie Pfad und Laufwerk des gegenwärtig in EXCEL eingestellten Verzeichnisses fest.
2. Wechseln Sie in das Verzeichnis, dessen Dateien beim Anzeigen von „GetOpenFilename“ zunächst gezeigt werden sollen.
3. Rufen Sie „GetOpenFilename“ auf.
4. Geben Sie bei Bedarf an, welche Art von Dateien angezeigt und ausgewählt werden dürfen.
5. Prüfen Sie, ob Dateien ausgewählt wurden und ob die Auswahl zulässig war.
   5.6 Falls ja, öffnen Sie die Datei(en).
   5.7 Falls nicht, informieren Sie den Benutzer und lassen Sie Ihr Programm entsprechend reagieren.
6. Wechseln Sie in das in Punkt 1. gesicherte Laufwerk und den gesicherten Pfad zurück.

Sie finden in der Arbeitsmappe „Öffnen.XLS“ im Modul „Mod GetOpenFilename“ ein ausprogrammiertes Beispiel, das alle diese Schritte enthält.

Aus dem von GetOpenFilename gelieferten vollen Pfad/Dateiname den Dateinamen extrahieren: Angenommen, „GetOpenFilename“ liefert zurück: “C:\Daten\Exclmak7\Dialog.XLS“, dann wenden Sie den folgenden Code an:

```
sDateiName = Dir (“C:\Daten\Exclmak7\Dialog.XLS“)
```

„sDateiName“ enthält dann „Dialog.XLS“.

Dateien kontrolliert speichern

Hier gilt sinngemäß das gleiche wie beim Öffnen. Sie können, mit Null Kontrolle über das, was Ihr User dann macht, den eingebauten Dialog „Speichern unter...“ aufrufen:

© by Josef Broukal, Vienna, Austria

```
Sub try()

Dim bWurdeGespeichert As Boolean

bWurdeGespeichert = Application.Dialogs(xlDialogSaveAs).Show
' bWurdeGespeichert ist True, falls Datei gespeichert wurde,
' False, wenn nicht
MsgBox bWurdeGespeichert

End Sub
```

Oder Sie benützen die Methode „GetSaveAsFilename“ mit der Syntax:

Application.GetSaveAsFilename(initialFilename, fileFilter, filterIndex, Title, buttonText)

Diese Methode gibt, falls Sie auf den Knopf „Speichern“ drücken, Namen und Pfad der zu speichernden Datei zurück. Brechen Sie den Dialog aber ab, erfolgt die Rückgabe False. Wird sie in einer Variablen vom Type „String“ aufgefangen, wandelt EXCEL False in den Text „Falsch“ um.

Nur **initialFilename** kommt bei „GetOpenFilename“ nicht vor. Es gibt Ihnen die Chance, einen Namen für die zu speichernde Datei vorzugeben. Dieser kann während des Programm-Laufes geändert werden.

Als Vorgangsweise bietet sich an:

1. Halten Sie Pfad und Laufwerk des gegenwärtig in EXCEL eingestellten Verzeichnisses fest.
2. Wechseln Sie in das Verzeichnis, dessen Dateien beim Anzeigen von „GetSaveAsFilename“ zunächst gezeigt werden sollen.
3. Rufen Sie „GetSaveAsFilename“ auf.
4. Prüfen Sie, ob das Dialogfeld mit der Anweisung, die Datei zu speichern, beendet wurde.
   4.5 Falls ja, speichern Sie die Datei
   4.6 Falls nicht, informieren Sie den Benutzer und lassen Sie Ihr Programm entsprechend reagieren.
5. Wechseln Sie in das in Punkt 1. gesicherte Laufwerk und dann in den gesicherten Pfad zurück.

Eine Anwendung vor unbefugtem Zugriff schützen

Was nützt das schönste Programm, die beste Datenanordnung, wenn die Menschen, die sie benützen, es in der Hand haben, sie mit einem Tastenanschlag, einem Mausklick zu zerstören? Wer VBA–Programme schreibt, kommt um das Schützen von Tabellen, Modulen und Arbeitsmappen nicht herum.

Den kompletten Code dieses Abschnittes finden Sie im Verzeichnis „Beispiel“ in der Datei „SCHÜTZEN.XLS“ im Blatt „Zellen“

Zellen

ZELLEN SCHÜTZEN

Wenn Sie ein neues Tabellenblatt erzeugen, dann sind alle Zellen „sperrbar“, aber der Zellschutz ist nicht eingeschaltet. Wollen Sie eine Tabelle zur Gänze schützen, brauchen Sie daher bloß zu schreiben:

© by Josef Broukal, Vienna, Austria

```
ActiveSheet.Protect
```

Wollen Sie den Schutz wieder aufheben, so geht das so:

```
ActiveSheet.Unprotect
```

Soll ein Teil der Zellen der Tabelle nach dem Schützen weiter bearbeitet werden können, so müssen Sie diese Zellen zunächst die Eigenschaft „Locked“ entziehen:

- Entziehen Sie allen Zellen die Eigenschaft „Gesperrt“
- Geben Sie den zu schützenden Zellen die Eigenschaft „Gesperrt“
- Schützen Sie das Blatt

```
Sub ZellenSchützen_1()

' Die nächste Anweisung ist nur für den Codetest gedacht
Sheets("Tabelle1").Activate
ActiveSheet.Unprotect
' Text zum Erkennen der zu sperrenden Zellen
ActiveSheet.Range(Cells(1, 1), Cells(10, 3)).Value = _
    "Gesperrt"
' Ganzes Blatt entsperren
ActiveSheet.Cells.Locked = False
' Zellen mit Text "Gesperrt" sperren
ActiveSheet.Range(Cells(1, 1), Cells(10, 3)).Locked = _
    True
' Blattschutz aktivieren
ActiveSheet.Protect
MsgBox "Die mit <<Gesperrt>> beschriebenen Zellen " & _
    "sind jetzt gesperrt. Bitte testen Sie das."

End Sub
```

ZELLEN WIEDER ÖFFNEN

Um alle Zellen eines Blattes wieder bearbeiten zu können, genügt der folgende Code:

```
Sub ZellenEntschützen()

Sheets("Tabelle1").Activate
ActiveSheet.Unprotect
MsgBox "Die mit <<Gesperrt>> beschriebenen Zellen " & _
    "sind jetzt wieder offen. Bitte testen Sie das."

End Sub
```

Blätter

BLATT SCHÜTZEN

Zum Schützen eines Tabellenblattes, VBA–Moduls, Diagramm oder Dialogblatt führt der folgende Code:

```
Sub TabelleSchützen()

Sheets("Tabelle1").Protect password:="MeinPasswort"

End Sub
```

© by Josef Broukal, Vienna, Austria

Password Ein Text, ohne dessen Angabe der Schutz nicht wieder aufgehoben werden kann. Wenn Sie dieses Argument weglassen, kann der Blattschutz von jedermann wieder aufgehoben werden.

Wahlweise kann der Blattschutz auch nur eingerichtet werden für:

- **Zeichnungsobjektliste** („drawingObjects“) Alles, was Sie mit Hilfe der Zeichenwerkzeuge über eine Tabelle gelegt haben, auch Schaltknöpfe. Bei Dialogblättern wird die Anordnung der Dialogelemente geschützt.
- **Inhalt** („contents“) Zellen können nicht mehr bearbeitet werden
- **Szenarioliste** („scenarios“) Die für die zu schützende Tabelle angefertigten Szenarios werden geschützt. Wird das Argument ausgelassen, werden die Szenarios ebenfalls geschützt.
- **Nur EXCEL 7.0:** Schutz gilt nicht für Änderung durch VBA-Programm („userInterfaceOnly“).

BLATTSCHUTZ WIEDER AUFHEBEN

```
Sub TabelleEntschützen()

Worksheets("Tabelle1").Unprotect password:="MeinPasswort"

End Sub
```

Den kompletten Code dieses Beispiels finden Sie im Verzeichnis „Beispiel“ in der Datei „SCHÜTZEN.XLS“ im Blatt „Blätter“

Arbeitsmappen

ARBEITSMAPPE SCHÜTZEN

Dazu führt der folgende Code:

```
Sub MappeSchützen()

ThisWorkbook.Protect Windows:=True, Structure:=True
MsgBox "Reihenfolge der Blätter kann nicht mehr geändert werden. " & _
    "Es können keine neuen Fenster erzeugt und keine bestehenden " & _
    "Fenster geschlossen werden. Bitte testen Sie das."

End Sub
```

Structure („Anordnung“) Wenn „Wahr“, können die Fenster der Arbeitsmappe nicht mehr verschoben werden.

Windows („Fenster“) Wenn „Wahr“, werden die Fenster der Arbeitsmappe geschützt.

Hinweis Anders als in der Online–Hilfe zu „Schützen“ dargestellt, führt das Ausführen der „Schützen“–Methode nicht zu einem Fehler, falls die Mappe schon geschützt ist.

MAPPE ENTSCHÜTZEN

```
DieseArbeitsmappe.Unprotect
```

Den kompletten Code dieses Beispiels finden Sie im Verzeichnis „Beispiel“ in der Datei „SCHÜTZEN.XLS“ im Blatt „Mappen“

Mehr zu diesem Thema in der Visual-Basic-Online-Hilfe unter „Methoden“, „Protect-Methode“

© by Josef Broukal, Vienna, Austria

Teile einer Anwendung ausblenden

Eine weitere Möglichkeit, Ihre Daten und Programme vor Fehlbedienung oder neugierigen Augen zu schützen, ist das Ausblenden. Dies wird—vom Namen über das Blatt bis zur Arbeitsmappe—durch das Setzen der „Visible“–Eigenschaft gesteuert.

DIE „VISIBLE“–EIGENSCHAFT

Sie ist „Wahr“, wenn das Objekt sichtbar ist. Bei einem Diagramm, Dialogblatt, Modul oder Tabellenblatt kann dieser Eigenschaft das benannte Argument „xlVeryHidden“ zugeordnet werden. Das Objekt wird ausgeblendet und kann nur sichtbar gemacht werden, indem diese Eigenschaft durch VBA–Code auf Wahr gesetzt wird. Der Anwender selbst kann das Objekt nicht einblenden.

Schreib-Lese-Zugriff.

Syntax

Objekt.Visible

Elemente

Objekt Erforderlich. Steht als Platzhalter des Objekts, dem diese Eigenschaft zugewiesen werden soll.

Anmerkungen

Die Visible-Eigenschaft für ein Element der Pivot-Tabelle ist Wahr, wenn das Element derzeit in der Tabelle angezeigt wird.

Wenn Sie der Visible-Eigenschaft für einen Namen den Wert Falsch zuweisen, erscheint der Name nicht im Dialogfeld „Namen festlegen“.

Den kompletten Code dieses Abschnittes finden Sie im Verzeichnis „Beispiel“ in der Datei „SICHTBAR.XLS“.

© by Josef Broukal, Vienna, Austria

Blätter

BLATT UNSICHTBAR MACHEN

Dazu dient der folgende Code:

```
ThisWorkbook.Sheets("Tabelle1").Visible = False
```

BLATT WIEDER ZEIGEN

```
ThisWorkbook.Sheets("Tabelle1").Visible = True
```

„SEHR“ UNSICHTBAR MACHEN

Die bisher gezeigten Beispiele entsprechen dem interaktiven Ausblenden von Blättern. Die Blätter können vom Benutzer über „Format“, „Blatt“ wieder auf den Schirm geholt werden.

Wollen Sie diese Möglichkeit ausschließen, dann müssen sie der „Visible“–Eigenschaft das benannte Argument „xlVeryHidden“ zuweisen. Blätter, die so verborgen werden, können nur noch mit einem VBA–Programm sichtbar gemacht werden:

```
ThisWorkbook.Sheets("Tabelle1").Visible = xlVeryHidden
```

Mappen

Genau genommen können Mappen nicht ausgeblendet werden, sondern lediglich die Fenster, in denen Sie eine Mappe zeigen. Wird eine Mappe in nur einem Fenster dargestellt, entspricht das Ausblenden des Fensters dem der Mappe. Von diesem Fall wollen wir in den folgenden Beispielen ausgehen.

MAPPE UNSICHTBAR MACHEN

```
ActiveWindow.Visible=False
```

oder genereller

```
Workbooks("SICHTBAR.XLS").Windows(1).Visible = False
```

MAPPE WIEDER SICHTBAR MACHEN

```
ThisWorkbook.Windows(1).Visible=True
```

oder genereller

```
Workbooks("SICHTBAR.XLS").Windows(1).Visible = True
```

Hinweis Zeilen und Spalten in einer Tabelle können Sie über die „Hidden“–Eigenschaft des Bereichs–Objektes unsichtbar machen. Formeln machen Sie über die „FormulaHidden“Eigenschaft unsichtbar.

Mehr zu diesem Thema in der Visual-Basic-Online-Hilfe unter „Eigenschaften“, „Visible-Eigenschaft“

© by Josef Broukal, Vienna, Austria

Eine Formel in eine Tabelle eintragen

Dazu gibt uns VBA die „Formula“–Eigenschaft zur Hand, mit der wir sowohl die Formel in einer Zelle erfragen als auch die Formel in einer oder mehreren Zellen festlegen (in die Zellen schreiben können).

Den kompletten Code dieses Abschnittes finden Sie im Verzeichnis „Beispiel“ in der Datei „FORMEL.XLS“.

FORMEL IN EINE ZELLE EINTRAGEN

```
Sub FormelInEineZelleEingeben()

ThisWorkbook.Sheets("Tabelle1").Cells(1, 2).Formula = _
    "=A1*5"

End Sub
```

FORMEL IN MEHRERE ZELLEN EINTRAGEN

```
Sub FormelInMehrereZellenEingeben()

ThisWorkbook.Sheets("Tabelle1"). _
    Range(Sheets("Tabelle1").Cells(1, 2), _
    Sheets("Tabelle1").Cells(5, 2)). _
    Formula = "=A1*5"

End Sub
```

Beachten Sie bitte, daß die Formel beim Eintragen an die relative Versetzung zur ersten Zelle, in der eingetragen wird, angepaßt wird. In der Tabelle steht dann

Zelle	Formel
A1	=A1*5
A2	=A2*5
A3	=A3*5…

Absoluter Bezug

Wenn Sie wollen, daß jede der fünf im obigen Beispiel eingetragenen Formeln sich auf die Zelle A1 bezieht, müssen Sie die Formel so schreiben:

"=$A$1*5"

Namen verwenden

Das Arbeiten mit „Namen“ ist eine der wesentlichen Stärken von EXCEL. Namen werden zwar meist dazu verwendet, um Zellbereiche in Tabellen zu beschreiben, aber Namen sind mehr: Sie sind Variable, denen Werte zugewiesen werden können. Dieser Wert kann eine Zelladresse sein, aber genausogut eine Zahl, ein Text, eine Rechenformel usw.

Namen werden in Arbeitsmappen gespeichert. Sie „gehören“ zur Arbeitsmappe, in der sie definiert sind. Namen sind Objekte; ihr Container–Objekt ist Names“.

Den kompletten Code dieses Abschnittes finden Sie im Verzeichnis „Beispiel“ in der Datei

© by Josef Broukal, Vienna, Austria

„NAMEN.XLS“.

Mit Angabe der Zellen Angenommen, Sie wollen die Zellen A1:A3 in „Tabelle1“ der aktiven Arbeitsmappe mit dem Namen „Namensbereich“ versehen. Dies geschieht im VBA–Code entweder so:

```
Sub NameFestlegen1()

  Sheets("Tabelle1").Select
  Range("A3:B5").Name = "Testbereich"
  MsgBox Range("Testbereich").Address(External:=True)

End Sub
```

oder so:

```
Sub NameFestlegen1a()

  ActiveWorkbook.Names.Add Name:="Namensbereich", _
    RefersToR1C1:="=Tabelle1!R1C1:Z3S1"

End Sub
```

Ausgewählte Zellen benennen Wenn Sie beim Programmieren noch gar nicht wissen können, welche Zellen mit einem Namen benannt werden sollen, müssen Sie so vorgehen:

```
Sub NameFestlegen2()

  ActiveWorkbook.Activate
  Sheets("Tabelle1").Select
  [A1].CurrentRegion.Activate
  ActiveWorkbook.Names.Add Name:="Arbeitsbereich", _
    RefersTo:="=" + Selection.Address
  MsgBox Range("Arbeitsbereich").Address(External:=True)

End Sub
```

In diesem Codebeispiel wechselt EXCEL tatsächlich in jenes Blatt, in dem der zu benennende Bereich liegt. Das können Sie vermeiden, indem Sie schreiben:

```
Sub NameFestlegen3()

  ActiveWorkbook.Names.Add Name:="Arbeitsbereich", _
    RefersTo:="=" + ActiveWorkbook.Sheets("Tabelle1"). _
    [A1].CurrentRegion.Address(External:=True)
  MsgBox Range("Arbeitsbereich").Address(External:=True)
End Sub
```

Zellen eines Zellbereichs–Objektes mit Namen benennen

© by Josef Broukal, Vienna, Austria

```
Sub NameFürBereichsObjektZellenBilden()

  Set BereichsObjekt = ActiveWorkbook.Sheets("Tabelle1"). _
    [A1].CurrentRegion
  ActiveWorkbook.Names.Add Name:="Namensbereich", _
    RefersTo:="=" + BereichsObjekt.Address(External:=True)
  MsgBox Range("Namensbereich").Address(External:=True)

End Sub
```

Aus benannten Zellen ein Bereichs–Objekt bilden

```
Sub ObjektAusNamensbereichBilden()

  Set BereichsObjekt = Evaluate(ActiveWorkbook. _
    Names("Namensbereich").Value)
  MsgBox BereichsObjekt.Address(External:=True)

End Sub
```

Zellen eines Bereichs–Objektes benennen

```
Sub NameFürBereichsObjektZellenBilden()

  Set BereichsObjekt = ActiveWorkbook.Sheets("Tabelle1"). _
    [A1].CurrentRegion
  ActiveWorkbook.Names.Add Name:="Namensbereich", _
    RefersTo:="=" + BereichsObjekt.Address(External:=True)
  MsgBox Range("Namensbereich").Address(External:=True)

End Sub
```

Namensbereich in den Vordergrund bringen Wenn Sie wollen, daß EXCEL die Tabelle und Zellen, auf die ein Name verweist, in den Vordergrund bringt (aktiviert), dann verwenden Sie dazu den folgenden Code:

```
Sub NamensbereichAnspringen1()

  Application.Goto Reference:= _
    ActiveWorkbook.Names("Namensbereich").Name

End Sub
```

Wenn Sie wollen, daß die linke obere Ecke des benannten Bereiches in die linke obere Ecke des Fensters gestellt wird, müssen Sie anordnen:

```
Sub NamensbereichAnspringen2()

  Application.Goto Reference:= _
    ActiveWorkbook.Names("Namensbereich").Name, _
    Scroll:=True

End Sub
```

Benannte Zellen mit Wert versehen Im ersten Beispiel wird der Bereich in den Vordergrund gebracht:

© by Josef Broukal, Vienna, Austria

```
Sub NamensbereichAusfüllen1()

  Application.Goto Reference:=ActiveWorkbook. _
    Names("Namensbereich").Name
  Selection.Value = "Ausgefüllt"

End Sub
```

Beispiel 2 stellt den Text „Ausgefüllt“ in die Zellen des benannten Bereiches, ohne zum Bereich zu wechseln:

```
Sub NamensbereichAusfüllen2()

  Evaluate(ActiveWorkbook.Names("Namensbereich").Value).Value = "Test"

End Sub
```

Um dieses Kunststück zusammenzubringen, müssen wir die „Evaluate“–Methode verwenden. Sie bildet aus dem Text, den

```
ActiveWorkbook.Names("Namensbereich").Value
```

zurückgibt („[Namen.XLS]Tabelle1!Z1S1:Z3S1“) ein Bereichs–Objekt. Und dem weisen wir, Zelle für Zelle, nochmals mit „Value“ die Zeichenfolge „Test“ zu.

© by Josef Broukal, Vienna, Austria

Daten filtern mit dem „Autofilter“

Der Autofilter, erreichbar über das Menü „Daten“ und den Befehl „Filter“, bietet eine sehr bequeme Möglichkeit zum Eingrenzen von Listen je nach dem Inhalt einzelner Zellspalten der Liste. Der Autofilter kann jedoch auch mit VBA gesteuert werden. Die dafür vorhandenen Eigenschaften und Methoden sind:

Eigenschaft/Methode	bewirkt/ergibt	Hinweis
<Range-Objekt>.Autofilter	kein Autofilter eingerichtet: richtet einen Autofilter für alle umliegenden beschriebenen Zellen ein. Autofilter vorhanden: löscht einen vorhandenen Autofilter.	Pro Tabellenblatt ist nur ein Autofilter möglich.
<Worksheet-Objekt>.AutoFilterMode	Gibt „True“ zurück, wenn im Blatt die Dorpdown-Pfeile für einen Autofilter angezeigt werden.	
<Worksheet-Objekt>.FilterMode	Gibt „True“ zurück, wenn im Blatt ein Autofilter eingerichtet wurde **und** es ausgefilterte Zeilen gibt. Gibt „False“ zurück, wenn im Tabellenblatt kein AutoFilter eingerichtet ist oder in der zu filternden Liste alle Zeilen eingeblendet sind.	
<Worksheet-Objekt>.ShowAllData	Blendet alle Zeilen für die momentan gefilterte Liste ein.	Versuch, diese Methode auf eine gar nicht gefilterte Liste anzuwenden, führt zu einer Fehlermeldung

Die Methode „Autofilter“

„Autofilter“ ist ein Schalter: Es richtet einen Autofilter ein, wenn es auf dem Blatt keinen gibt, und falls es einen gibt, schaltet es ihn aus. „Autofilter“ führt zu einer Fehlermeldung, falls der Cursor in einer Zelle steht, die leer ist. Der Autofilter wird übrigens nicht in der Zelle eingerichtet, in der der Cursor steht. EXCEL sucht alle um diese Zelle liegenden beschriebenen Zellen ab und setzt die Filter-Pfeile dann in der obersten Zeile des Bereichs.

© by Josef Broukal, Vienna, Austria

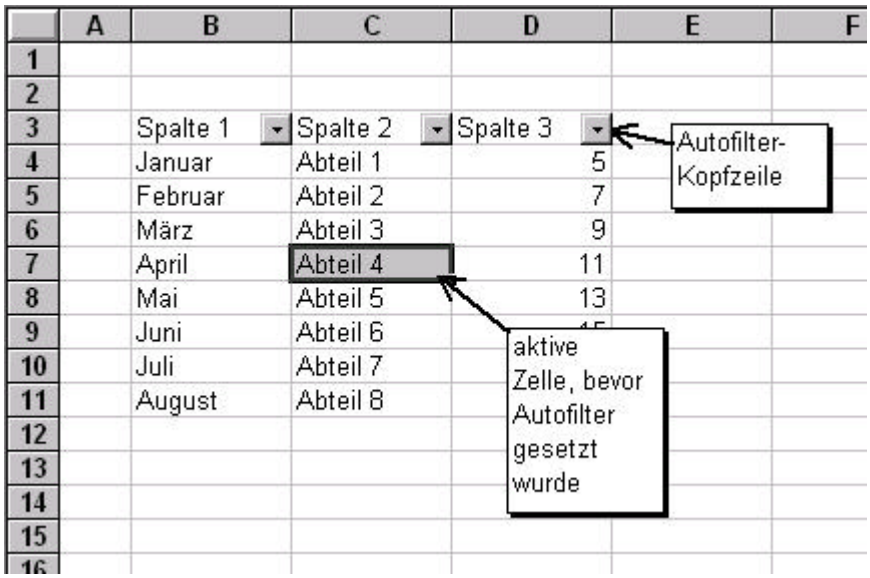


Abbildung 35: Die „Autofilter“-Methode ist intelligent. Sie findet von selbst die oberste Zeile eines Listenbereiches und stellt dorthin die „Autofilter“-Symbole

```
Sub AutofilterToggle()

' Autofilter gibt's nur in Tabellenblättern:
If TypeName(ActiveSheet) <> "Worksheet" Then
  Beep
  Exit Sub
' Falls kein Autofilter gesetzt ist, muß die aktive Zelle
' einen Wert haben, damit ein Autofilter gesetzt werden kann:
ElseIf ActiveSheet.AutoFilterMode = False And IsEmpty(ActiveCell) Then
  MsgBox "Autofilter kann nicht in einer leeren Zelle " & _
    "eingerichtet werden !", vbCritical, "Fehler:"
  Exit Sub
End If

' Falls Autofilter gesetzt, Filter löschen; falls nicht
' gesetzt, Filter einrichten:
ActiveCell.AutoFilter

End Sub
```

Hinweis Der Versuch, einen Autofilter auf einer anderen Blattart als „Worksheet“ einzurichten, führt ebenso zu einer Fehlermeldung wie der Versuch, einen Autofilter einzurichten, während der Cursor in einer leeren Tabellenzelle steht.

Hinweis Es gibt keine Möglichkeit, mit VBA-Code die Filterbegriffe eines gesetzten Autofilters zu erfragen.

Die Eigenschaft „AutoFilterMode“

Diese Eigenschaft hat den Wert „True“, wenn die Dropdown-Pfeile momentan angezeigt werden. Sie können diese Eigenschaft auf „False“ setzen, um die Pfeile zu entfernen, jedoch nicht auf „True“, um sie einzublenden. Verwenden Sie hierzu die „AutoFilter“-Methode.

© by Josef Broukal, Vienna, Austria

Die Eigenschaft „FilterMode“

Sie kann nur befragt werden. Setzen kann man sie nicht. „FilterMode“ ergibt „True“, falls es 1.) auf dem Blatt einen Autofilter gibt **und** dieser 2.) gefilterte Zeilen enthält. Man kann diese Eigenschaft nur für einen Zweck verwenden: Zu prüfen, ob es einen Sinn macht, per Code anzuordnen, daß wieder alle Zeilen der gefilterten Liste gezeigt werden sollen. Wird das nämlich angeordnet, ohne daß es gefilterte Zeilen gibt, reagiert EXCEL mit einer Fehlermeldung.

```
Sub AlleDatenZeigen()

If TypeName(ActiveSheet) <> "Worksheet" Then
  Beep
ElseIf ActiveSheet.FilterMode = False Then
  Beep
Else
  ActiveSheet.ShowAllData
End If

End Sub
```

Daten filtern

Verwenden Sie dazu die „Autofilter“-Methode in folgender Syntax:

```
<Tabellenbereich>.AutoFilter(field, criteria1, operator, criteria2)
```

field die fortlaufende Nummer der zu filternden Spalte im Autofilter-Bereich.

criteria1 ein Filterbegriff, z.B. “=Müller“

operator „xlAnd“ oder „xlOr“ für Verknüpfung mit „criteria2“; „xlTop10Items“ gibt jene Anzahl an Datenzeilen zurück, die in „criteria1“ mit einer Zahl angegeben werden. Zum Beispiel: „10“:

```
Sub Makro2()
  Selection.AutoFilter Field:=1, Criteria1:="10", Operator:= _
    xlTop10Items
End Sub
```

criteria2 zweiter Filterbegriff, z.B. “=Huber“

Hinweis Beachten Sie, daß zusätzlich zu den Filterbegriffen auch die Vergleichs-Operatoren in einer Zeichenkette an die „Autofilter“-Methode übergeben werden müssen.

„AUTOFILTER“-VERGLEICHSOPERATOREN

Alle Operatoren können auf Texte, Datum und Zahlen angewendet werden. Die Umlaute werden bei deutscher Ländereinstellung von Windows korrekt behandelt. Groß- und Kleinschreibung sind gleichgesetzt.

Vergleichs-Operator	bewirkt	Beispiel
=	Es werden nur jene Datenzeilen gezeigt, in denen in der Spalte „field“ der Filterbegriff komplett enthalten ist	=Huber =123,12 =01.04.96

© by Josef Broukal, Vienna, Austria

>	Es werden nur jene Datenzeilen gezeigt, deren Wert in der Spalte „field“ größer ist als im Filterbegriff angegeben.	>123,12 >Hofer
<	Es werden nur jene Datenzeilen gezeigt, deren Wert in der Spalte „field“ kleiner ist als im Filterbegriff angegeben.	<01.04.1996 <123,12 <Österreich
>=	Es werden nur jene Datenzeilen gezeigt, deren Wert in der Spalte „field“ größer ist als im Filterbegriff angegeben.	>=01.01.96 >=123,12 >=Österreich
<=	Es werden nur jene Datenzeilen gezeigt, deren Wert in der Spalte „field“ kleiner ist als im Filterbegriff angegeben.	<=01.01.96 <=123,12 <=Österreich
<>	Es werden nur jene Datenzeilen gezeigt, deren Wert in der Spalte „field“ nicht dem im Filterbegriff angegebenen Wert entspricht.	<>01.01.96 <>123,12 <>Österreich

Hinweis Beachten Sie, daß zusätzlich zu den Filterbegriffen auch die Vergleichs-Operatoren in der Zeichenkette mit dem Filterbegriff an die „Autofilter“-Methode übergeben werden müssen:

```
Sheets("Tabelle1").Cells(1, 1).AutoFilter _
  field:=1, _
  criteria1:=">Huber"
```

Autofilter löschen Wollen Sie den Filter über eine Datenspalte löschen, dann müssen Sie die „Autofilter“-Methode „leer“ aufrufen:

```
ActiveSheet.Cells(1, 1).AutoFilter _
    field:=NAMENFELD
```

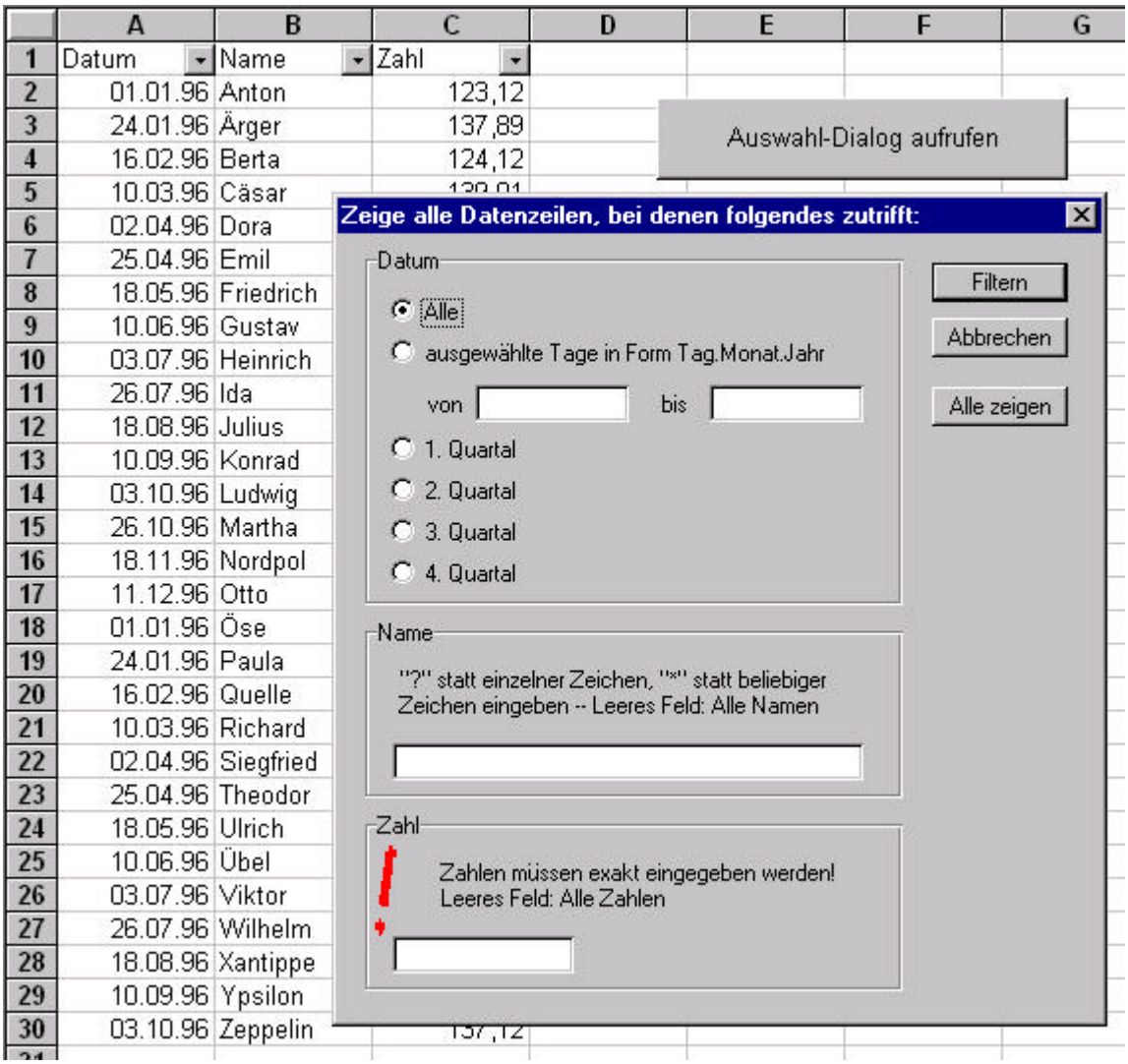
AUSPROGRAMMIERTES BEISPIEL

Um die Funktionalität und die Unzulänglichkeiten des „Autofilters“ zu zeigen, verwenden wir eine dreispaltige Liste:

- Spalte 1 enthält Datumswerte
- Spalte2 enthält Textwerte
- Spalte3 enthält Zahlen

Liste und Dialogfeld für das Filtern sehen so aus:

© by Josef Broukal, Vienna, Austria



	A	B	C
1	Datum	Name	Zahl
2	01.01.96	Anton	123,12
3	24.01.96	Ärger	137,89
4	16.02.96	Berta	124,12
5	10.03.96	Cäsar	
6	02.04.96	Dora	
7	25.04.96	Emil	
8	18.05.96	Friedrich	
9	10.06.96	Gustav	
10	03.07.96	Heinrich	
11	26.07.96	Ida	
12	18.08.96	Julius	
13	10.09.96	Konrad	
14	03.10.96	Ludwig	
15	26.10.96	Martha	
16	18.11.96	Nordpol	
17	11.12.96	Otto	
18	01.01.96	Öse	
19	24.01.96	Paula	
20	16.02.96	Quelle	
21	10.03.96	Richard	
22	02.04.96	Siegfried	
23	25.04.96	Theodor	
24	18.05.96	Ulrich	
25	10.06.96	Übel	
26	03.07.96	Viktor	
27	26.07.96	Wilhelm	
28	18.08.96	Xantippe	
29	10.09.96	Ypsilon	
30	03.10.96	Zeppelin	

Den kompletten Code dieses Beispiels finden Sie im Verzeichnis „Autofilt“ in der Datei „AUTOFILT.XLS“.

Datum und Zahlen—zwei knifflige Probleme

Wollen wir ein Datum filtern und ordnen im VBA-Code an, daß der deutsche Datumstext (z.B. „03.07.96“ als Filterbegriff verwendet werden soll, wird die gesamte Liste ausgefiltert. EXCEL erwartet offensichtlich, daß das Datum im Format der VBA-Standardsprache, Englisch/USA, zur Verfügung gestellt wird. In unserem Fall: „07/03/96“. Leider nützt es auch nichts, im Eingabefeld einer Dialogbox dieses Datumsformat anzuwenden. Bevor VBA und EXCEL es als Filterbegriff verwenden, wird es automatisch in ein „deutsches“ Datum umgewandelt. Die einzige Abhilfe, die ich derzeit sehe, besteht im Anfertigen einer Zeichenkette, die das US-Datumsformat enthält. Für die Mitteilung besserer Wege bin ich dankbar.

```
Function SystemDatumFormat(sDasDatum As String)
SystemDatumFormat = Month(sDasDatum) & "/" _
  & Day(sDasDatum) & "/" & Year(sDasDatum)

End Function
```

Bei Zahlen-Eingabe führt das im Deutschen übliche Dezimal-Komma dazu, daß die gesuchte Zahl nicht gefunden und die ganze Liste ausgefiltert wird. Die Eingabe eines Dezimalpunktes löst das

© by Josef Broukal, Vienna, Austria

Problem nur für den Wissenden. Besser ist es, im VBA-Code den Punkt gegen das Komma so zu tauschen:

```
Function SystemFormatZahl(sDieZahl As String)

SystemFormatZahl = Application.Substitute(sDieZahl, ",", ".")

End Function
```

TEXTE MIT WILDCARDS FILTERN

Datum und Zahlen können nur komplett gesucht werden, bei Texten können „?“ und “*“ verwendet werden, um Zeilen herauszufiltern, deren Text dem eingegebenen Filterbegriff nur teilweise entspricht:

“Hu*“ findet zum Beispiel jeden Text, der mit den Buchstaben „Hu“ beginnt, “*Hu*“ findet „Hu“ (auch kleingeschrieben) an jeder Stelle in einem Text. “Hu?“ findet jeden Text, der mit „Hu“ beginnt und dann noch einen dritten Buchstaben hat.

© by Josef Broukal, Vienna, Austria

Menüs mit VBA verändern

In diesem Abschnitt lernen Sie:

- eingebaute EXCEL–Menüs ergänzen und verkürzen
- eigene Menüleisten mit eigenen Menüs und Befehlen erzeugen

Den kompletten Code der Beispiele in diesem Kapitel finden Sie im Verzeichnis „Beispiel“ in der Datei „MENÜS.XLS“.

VBA gibt Ihnen die Mittel an die Hand, auf Ihre Anwendung maßgeschneiderte Menüs herzustellen, die den Programm–Benützer auf jene Arbeitsmöglichkeiten beschränken, die Sie vorsehen.

EXCEL hat neun eingebaute Menüleisten: für Tabellen, Diagramme, VBA–Module, Info–Fenster und—damit Sie alte Programme weiter verwenden können—spezielle Menüleisten von EXCEL 3.0 und EXCEL 4.0. Dazu kommen noch 25 „Kontextmenüs“, die Sie durch Drücken der rechten Maustaste aufklappen können. Je nachdem, ob eine Tabelle, ein Diagramm oder ein VBA–Programm im Vordergrund ist, enthalten sowohl Menüleisten wie auch Kontextmenüs verschiedene Menüs und verschiedene Befehle.

Mehr zu die Namen der Menüleisten bzw. der Kontextmenüs in der Visual-Basic-Online-Hilfe unter „Methoden“, „Menubars–Methode“ bzw. „ShortcutMenus–Methode“

Hinweis Anstelle mit einem Makro können Sie benutzerdefinierte Menüleisten auch auf einfache Weise mit dem Menü-Editor erstellen. Eine schrittweise Anleitung dazu finden Sie im Visual Basic Benutzerhandbuch

Befehle

Einen eingebauten Befehl löschen

Ich will Ihnen anhand des Löschens des Befehls „Datei-Manager...“ im Menü „Datei“ der Datei–Menüleiste zeigen, wie Sie vorgehen müssen, wenn Sie einen eingebauten Befehl löschen wollen.

```
Sub EingebautenBefehlLöschen()

MenuBars(xlWorksheet).Menus.Item("Datei"). _
    MenuItems("Datei-Manager...").Delete

End Sub
```

Wie Sie sehen, müssen wir uns—schön objektorientiert—von der Menüleiste über das Menü bis zum Befehl vorarbeiten. Jedes diese Objekte ist als Bestandteil einer Objektauflistung anzusprechen. Am Ende lösen wir dann die Löschen–Methode aus.

Die verschiedenen Menüleisten sprechen Sie über symbolische Konstante an:

Konstante	Beschreibung

© by Josef Broukal, Vienna, Austria

Konstante	Beschreibung
xlChart	Diagrammblatt
xlChart4	Microsoft Excel 4-Diagramm-Menüleiste
xlChartShort	Kurzes Microsoft Excel 3-Diagramm-Menü
xlInfo	Infofenster
xlNoDocuments	Keine geöffneten Dateien
xlModule	Visual Basic-Modul
xlWorksheet	Tabellenblatt, Makroblatt und Dialogblatt
xlWorksheet4	Microsoft Excel 4-Tabellenblatt-Menüleiste
xlWorksheetShort	Kurzes Microsoft Excel 3-Tabellenblatt-Menü

Einen eingebauten Befehl wiederherstellen

```
Sub EingebautenBefehlWiederherstellen_1()

MenuBars(xlWorksheet).Menus("Datei"). _
  MenuItems.Add _
    Caption:="Datei-Manager...", _
    Restore:=True, _
    Before:="Datei-Info..."

End Sub
```

Nach Angabe von Menüleiste und Menü fügen wir mit der „Add“–Methode ein Menüelement dazu und geben an:

Caption	Den Namen des Menüelementes einschließlich der Punkte, wo vorhanden
Restore	weist EXCEL an, die Funktionalität des einzufügenden eingebauten Menüelements wieder zur Verfügung zu stellen
Before	Das Menüelement, vor dem das neue Element in das Menü eingefügt werden soll. Lassen Sie dieses Argument weg, kommt das neue Element ans Ende des Menüs.

Einen eigenen Befehl einfügen

Beachten Sie bitte, daß Sie sich wieder für eine der neun eingebauten Menüleisten und für ein Menü innerhalb der gewählten Leiste entscheiden müssen, bevor Sie einen selbstdefinierten Befehl einfügen können. Mit diesem Befehl werden Sie in der Regel ein VBA–Programm aufrufen, das Sie natürlich auch selbst schreiben müssen.

© by Josef Broukal, Vienna, Austria

```
Sub EigenerBefehlEinfügen()

MenuBars(xlModule).Menus("Ansicht"). _
  MenuItems.Add _
    Caption:="Wie spät ist es?", _
    OnAction:="WieSpätIstEs", _
    Before:=1

End Sub


Sub WieSpätIstEs()

MsgBox "Es ist jetzt genau: " & Now

End Sub
```

Wie Sie sehen, braucht es zum Einfügen eines eigenen Befehls nicht nur die benannten Argumente „Caption“ und „Before“, sondern auch „OnAction“, dem Sie den Namen des VBA–Programmes zuweisen, das bei der Wahl des Menüelementes ausgeführt werden soll.

Einen eigenen Befehl löschen

Wieder müssen wir, von der Ebene der Menüleisten weg bis zum Menüelement buchstabengetreu angeben, was wir löschen wollen. Stehen hinter dem Text des Menüelementes drei Punkte(z.B. „Datei-Manager...“), so sind auch diese anzugeben.

Nach vollständiger Definition des gemeinten Menüelementes wenden wir die „Löschen“–Methode auf das Element an.

```
Sub EigenerBefehlLöschen()

MenuBars(xlModule).Menus("Ansicht"). _
    MenuItems("Wie spät ist es?").Delete

End Sub
```

Zwischenstrich einfügen

Ein Zwischenstrich ist ein Menüelement wie jedes andere. Er wird „hinzugefügt“ und „gelöscht“. Definiert wird er durch einen Bindestrich. Natürlich kann ein Zwischenstrich keine Aktion auslösen. Diese Eigenschaft hat er also nicht. Ansonsten aber programmieren wir wie gehabt:

© by Josef Broukal, Vienna, Austria

```
Sub ZwischenstrichEinfügen()

MenuBars(xlModule).Menus("Ansicht"). _
  MenuItems.Add _
    Caption:="-", _
    Before:=2

End Sub
```

Zwischenstrich löschen

Beim Löschen eines Zwischenstriches haben wir das Problem, daß es in den meisten Menüs mehr als einen davon gibt. Die Angabe des Bindesstriches definiert in so einem Fall daher nicht ausreichend genau, welchen Bindestrich wir meinen. Daher ist anstelle der Aufschrift die Ziffer für die Position des zu löschenden Zwischenstriches anzugeben:

```
Sub ZwischenstrichLöschen()

MenuBars(xlModule).Menus("Ansicht"). _
MenuItems(2).Delete

End Sub
```

Menüs

Menü in Menüleiste einfügen

Wie bei Befehlen müssen wir auch hier die Menüleiste angeben, in die das neue Menü eingefügt werden soll. Vom neuen Menü, das der Menüliste der Menüleiste hinzugefügt wird, müssen wir den Namen angeben, optional auch die Stelle, an der es stehen soll.

```
Sub MenüEinfügen()

MenuBars(xlModule).Menus.Add _
    Caption:="Mein Menü", _
    Before:=3

End Sub
```

Hinweis Anstelle der Zahl „3“ können wir auch den Namen des Menüs hinschreiben, das derzeit die dritte Stelle in der Menüleiste einnimmt: „Ansicht“.

© by Josef Broukal, Vienna, Austria

Menü löschen

```
Sub MenüLöschen()

MenuBars(xlModule).Menus("Mein Menü").Delete

End Sub
```

Untermenü einfügen

Damit Sie einem Menübefehl weitere Untermenüs zuordnen können, muß der Menübefehl anstelle von "MenüelementListe.Hinzufügen" mit "MenüelementListe.MenüHinzufügen" erstellt werden. Auf diese Weise erscheint rechts vom Menübefehl der schwarze Pfeil, der weitere Untermenüs anzeigt. Davon ausgehend können die weiteren Untermenüs erstellt werden.

Nachfolgendes Beispiel erstellt die Menüleiste "MeineLeiste" mit dem Menü "MeinMenü", das zwei Menübefehle enthält. Der Menübefehl "Menüpunkt1" hat zwei weitere Untermenüs. Mit dem Menübefehl "ExcelMenü" wird das Makro "zurueck" aufgerufen, das die Excel Menüleiste für Module wieder einblendet:

```
Sub LeisteErstellen()

Set mleiste = MenuBars.Add("MeineLeiste")
Set mpunkt = mleiste.Menus.Add("MeinMenü")
Set mbefehl = mpunkt.MenuItems.AddMenu("Menüpunkt1")
mbefehl.MenuItems.Add Caption:="Untermenü1"
mbefehl.MenuItems.Add Caption:="Untermenü2"
mpunkt.MenuItems.Add Caption:="ExcelMenü", _
OnAction:="zurueck"
mleiste.Activate

End Sub
```

Excel Menü wieder einblenden:

```
Sub zurueck()

 Application.MenuBars(xlModule).Activate
 MenuBars("MeineLeiste").Delete

 End Sub
```

Kontextmenüs

Kontextmenü erweitern

Auch die mit der rechten Maustaste auf den Schirm zu bringenen Kontextmenüs für Tabellen, Visual–Basic Module, Diagramme und andere Objekte von EXCEL sind durch ein VBA–Programm veränderbar. Eine Auflistung der symbolischen Konstanten, mit denen Sie jenes Kontextmenü ansprechen, das Sie verändern wollen, finden Sie in der VBA–Hilfe unter „ShortcutMenus–Methode“. Die wichtigsten sind:“xlWorksheet“, “xlModule“, “xlNoDocuments“ und “xlCharts“.

© by Josef Broukal, Vienna, Austria

```
Sub KontextmenüelementHinzufügen()

Application.ShortcutMenus(xlModule).MenuItems.Add _
    Caption:="Testeintrag", _
    OnAction:="WieSpätIstEs", _
    Before:=1

End Sub
```

Kontextmenü löschen

Hier ist ein kurzer Beispielscode:

```
Sub KontextmenüelementLöschen()

Application.ShortcutMenus(xlModule).MenuItems("Testeintrag").Delete

End Sub
```

Ausprogrammiertes Beispiel

Nachdem Sie jetzt mehrere Beispiele für das Einfügen und Löschen von eingebauten und selbstdefinierten Befehlen kennengelernt haben, möchte ich mit Ihnen ein umfangreicheres, durchprogrammiertes Beispiel erarbeiten. Wir wollen

- eine neue Menüleiste einrichten;
- in dieser Menüleiste ein Menü „Mein Menü“ schaffen;
- dort mehrere Befehle einfügen;
- einen Befehl wählbar und nicht wählbar machen;
- vor einen Befehl ein Häkchen stellen, das zeigt, ob der Befehl aktiviert worden ist,
- die neue Menüleiste wieder löschen.

Code

```
Private NeueMenüleiste As MenuBar
Private EinMenü As Menu
Private DefBereich As Range
Private EineDefZeile As Range

Sub NeueLeisteAufbauen()

Rem Neue Menüleiste durch Einfügen eines neuen Elementes an die
Rem MenüleisteListe-Aufzählung als Objekt schaffen.


Set NeueMenüleiste = MenuBars.Add

Rem In neuer Menüleiste das Menu "Mein Menü" einrichten

NeueMenüleiste.Menus.Add _
Caption:="Mein Menü"

Rem In das neue Menü Befehle einfügen und mit den Angaben
Rem für "Aufschrift" und "BeiAktion" versehen.
Rem Diese Angaben verwalten wir In Tabelle1 dieser Arbeitsmappe
Rem In einem Bereich, dessen obere linke Ecke in Zelle A1 liegt
```

© by Josef Broukal, Vienna, Austria

```
Rem Range ohne Titelzeile als Objekt festhalten

Set DefBereich = ThisWorkbook.Sheets("Tabelle1"). _
    [a1].CurrentRegion
Set DefBereich = DefBereich. _
    Resize(DefBereich.Rows.Count - 1). _
    Offset(1, 0)

Rem Mit Hilfe der Zeileliste-Methode lesen wir aus dem Bereich
Rem jetzt die Angaben für Aufschrift" And "BeiAktion" aus

For Each EineDefZeile In DefBereich.Rows
    NeueMenüleiste.Menus("Mein Menü").MenuItems. _
    Add _
    Caption:=EineDefZeile.Columns(1), _
    OnAction:=EineDefZeile.Columns(2)
Next

Rem Und jetzt zeigen wir die neue Menüleiste!

NeueMenüleiste.Activate

End Sub


Rem VBA-PROGRAMME ZUM AUSFÜHREN DER BEFEHLE--------------------

Sub Uhrzeit()

MsgBox "Die Uhrzeit In Ihrem Computer ist: " & _
    Format(Now, "hh:mm")

End Sub


Sub Arbeitsmappenname()

MsgBox "Diese Arbeitsmappe heißt: " & ThisWorkbook.Name

End Sub


Sub UhrzeitBefehlManipulieren()

Rem Um Schreibarbeit zu sparen And klareren Code zu erhalten,
Rem weisen wir die immer wiederkehrende Ansprache: Name der Menü-
Rem leiste, Name des Menüs, ein Element des Menüs, einer
Rem Objektvariablen "derBefehl" zu.

Dim derBefehl As Object

Set derBefehl = NeueMenüleiste.Menus("Mein Menü"). _
    MenuItems
If derBefehl("Wie spät ist es?").Enabled = True Then
    derBefehl("Wie spät ist es?").Enabled = False
    derBefehl("Uhrzeit-Befehl abblenden"). _
        Caption = "Uhrzeit-Befehl aufblenden"
Else
```

© by Josef Broukal, Vienna, Austria

```
    derBefehl("Wie spät ist es?").Enabled = True
    derBefehl("Uhrzeit-Befehl aufblenden"). _
        Caption = "Uhrzeit-Befehl abblenden"
End If

End Sub


Sub HäkchenManipulieren()

With NeueMenüleiste.Menus("Mein Menü"). _
    MenuItems("Häkchen manipulieren")
        .Checked = _
        Not .Checked
End With

End Sub



Sub AlteLeisteZeigen()

NeueMenüleiste.Delete
MenuBars(xlModule).Activate

End Sub

Sub auto_close()

If ActiveMenuBar.Index = NeueMenüleiste.Index Then
  AlteLeisteZeigen
End If

End Sub
```

© by Josef Broukal, Vienna, Austria

Selbstgestaltete Dialogfelder

Wann immer…

- …Sie sich nicht darauf verlassen wollen, daß ein Benützer einer EXCEL-Anwendung alles weiß und beherrscht, was zum korrekten Umgang mit der Anwendung notwendig ist
- …Sie vor dem Eintragen von Daten abprüfen wollen, ob diese plausibel sind
- …Berechnungen durchzuführen sind und nur deren Ergebnisse in Tabellen festgehalten werden sollen
- …Eingaben sich zum Teil wiederholen

brauchen Sie ein selbstgestaltetes Dialogfeld.

EXCEL verwaltet diese Dialogfelder in „Dialogblättern“ von Arbeitsmappen. Diese „Dialogblätter“ stellen eine Zeichenfläche dar, auf der sie die Anordnung der Dialogelemente entwerfen und die Verhaltensweisen der einzelnen Elemente festlegen.

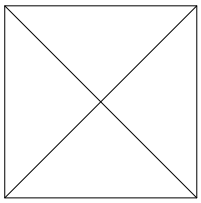


Abbildung 36: eine leere Zeichnungsfläche und die Symbolleiste für selbstgestaltete Dialoge

Dialogblatt anfordern Verwenden Sie dazu den Befehl EINFÜGEN, MAKRO, DIALOG.

Die Symbolleiste „Dialog“

Diese Symbolleiste wird automatisch eingeblendet, sobald sie ein Dialogblatt anlegen oder zu ihm wechseln. Sie enthält folgende Elemente:

Textfeld („Textbox“)	Bearbeitungsfeld („Editbox“)
Gruppenfeld („Groupbox“)	Schaltfläche („Button“)
Kontrollkästchen („Checkbox“)	Optionsfeld („Optionbutton“)
Listenfeld („Listbox“)	Dropdown-Listenfeld („Dropdown“)
Kombination Text—Listenfeld („Listbox“ und „Editbox“)	Dropdown Kombination Textfeld—Listenfeld („Dropdown“ und „Editbox“)
Laufleiste („Scrollbar“)	Drehfeld („Spinner“)
Eigenschaften	Code bearbeiten
Zeichenraster	Dialog ausführen

Abbildung 37: Elemente der Dialogleiste „Symbol“ und ihre Bedeutung

© by Josef Broukal, Vienna, Austria

Elemente eines Dialogfeldes

Jedes Dialogfeld setzt sich aus folgenden Elementen zusammen:

- dem Dialogfeldrahmen einschließlich des Dialogfeld-Titels
- den eigentlichen Dialogelementen (Textflächen, Eingabe- und Auswahl-Elemente, Schaltknöpfe)

Rahmen, Titel und Elemente können bereits vor dem Anzeigen des Dialogs angesprochen werden. Sie können mittels VBA-Code die Größe des Dialogfeldrahmens festlegen, in Bearbeitungsfelder Text hineinstellen, Optionsfelder und Kontrollkästchen an- und abschalten, Listenfeldern die Listenwerte zuweisen, usw.:

Der lange Weg zum fertigen Dialogfeld...

Dialogfelder mit mehr als ein paar Texteingabefeldern herzustellen, ist eine langwierige Arbeit; es ist dafür viel Code zu schreiben. Die einzelnen Phasen sind:

1. Zeichnen des Dialogfeldes
2. Reihenfolge der Elemente festlegen
3. Code für das Füllen der Elemente vor dem Anzeigen des Dialogfeldes schreiben
4. Code für die laufende Kontrolle des Dialoges schreiben
5. Code für die Kontrolle der Eingaben vor der Übernahme in die Tabelle schreiben

Zeichnen des Dialogfeldes

Dialogelement einfügen Dies ist eine Arbeit, die Sie nur mit der Maus durchführen können! (Der Vollständigkeit halber füge ich hinzu, auch mit VBA-Befehlen, aber warum so umständlich?) Gehen Sie dazu so vor:

- Klicken Sie mit der Maus auf das Symbol für das Dialogelement, das Sie in die Zeichenfläche setzen wollen. Der Cursor nimmt daraufhin die Form eines Kreuzes an
- Setzen Sie den Cursor in der Zeichenfläche an jene Stelle, an der die linke obere Ecke des Dialogelementes zu liegen kommen soll.
- Drücken Sie die linke Maustaste. Ziehen Sie die Maus mit gedrückter Taste nach rechts unten. Ein strichlierter Rahmen zeigt Ihnen, wie groß das Element sein wird, sobald Sie die Maustaste wieder auslassen.
- Lassen Sie die Maustaste los, sobald Sie mit der Größe des Dialogelementes zufrieden sind.

Mehrere gleichartige Elemente in einem einfügen Setzen Sie den Cursor auf den Rand des ersten eingefügten Elementes. Drücken Sie dann die Strg–Taste. EXCEL stellt daraufhin eine Kopie des Elementes zur Verfügung (sie liegt direkt über dem Element, ist also zunächst nicht sichtbar; der Mauszeiger erhält aber zusätzlich ein kleines Kreuz). Ziehen Sie die Kopie dorthin, wo Sie sie haben wollen. Tip: Wenn Sie zusätzlich auch die Umschalt–Taste drücken, ist nur waagrechte oder senkrechte Bewegung möglich (je nachdem, wie Sie die Bewegung beginnen).

Dialogelement verschieben Klicken Sie dazu mit der linken Maustaste auf das Element. Sobald der Rahmen erscheint, setzen Sie den Mauszeiger auf diesen. Ziehen Sie dann mit gedrückter linker Maustaste das Element an die gewünschte Stelle.

Größe ändern Der Rahmen um ein Dialogelement enthält 8 Anfasser (kleine Rechtecke). Setzen Sie den Mauscursor auf einen Anfasser, drücken Sie die linke Maustaste und ziehen Sie die Maus dorthin, wo der Anfasser zu liegen kommen soll.

© by Josef Broukal, Vienna, Austria

Dialogelement löschen Klicken Sie mit der rechten Maustaste auf das Element. Wählen Sie im Kontextmenü den Befehl „Objekt löschen“.

Mehrere Dialogelemente zugleich bearbeiten Klicken Sie mit der Maus außerhalb des Dialogfeldes auf das Dialog-Arbeitsblatt, damit kein Element ausgewählt ist. Ziehen Sie dann mit gedrückter linker Maustaste eine Linie um die Elemente, die Sie zugleich bearbeiten wollen. Sobald Sie die Maustaste auslassen, erscheinen alle von der Linie umschlossenen Elemente markiert und können gemeinsam verschoben, vergrößert/verkleinert oder gelöscht werden.

Mehrere nicht nebeneinander liegende Elemente werden ausgewählt, indem das erste angeklickt und alle weiteren bei gedrückter Umschalt-Taste angeklickt werden.

NAME DES DIALOGELEMENTES FESTLEGEN

Wenn Sie den Wert eines Dialogelementes auslesen oder festsetzen wollen, müssen Sie es mit seinem Namen angeben. EXCEL gibt automatisch einen Namen vor, wenn Sie das Element schaffen. Dieser Name besteht aus zwei Teilen:

1. Der Typ des Elementes („Bearbeitungsfeld“, „Listenfeld“ usw.)
2. Eine fortlaufende Ziffer

Diese Namen sind natürlich nicht aussagekräftig. Was sagt Ihnen schon beim Schreiben von Code z.B. „Listenfeld 24“?

Daher sollten Sie diese vorgegebenen Namen sofort nach dem Einfügen des Elementes ersetzen. Klicken Sie dazu auf das Feld links in der Bearbeitungszeile (dort würde in unserem Beispiel „Listenfeld 24“ stehen). EXCEL markiert dann den gesamten Namen. Schreiben Sie jetzt den neuen Namen **und drücken Sie unbedingt die Eingabetaste**. Nur dann wird der Name auch wirklich akzeptiert.

Namenskonventionen Vor allem bei größeren Programmen ist es wichtig, den Überblick zu bewahren. Daher bietet sich folgende Konvention an:

1. Kurzbezeichnung des Dialogs (z.B. „EDlg“ für „Eingabedialog“)
2. Ein Unterstrich („_“)
3. Kurzbezeichnung für die Art des Elementes (z.B. „EB“ für „Editbox“)
4. Ein Unterstrich („_“)
5. Ein sprechender Name für das, was in dem Element erfragt werden soll (z.B. „FamName“ für ein Bearbeitungsfeld, in dem ein Familienname eingegeben werden soll)

Der Name des Elementes würde in unserem Beispiel also lauten: „EDlg_EB_Famname“

Reihenfolge der Elemente festlegen

In jedem Dialogfeld gibt es eine natürliche Abfolge der Eingaben—wahrscheinlich zeilenweise von oben nach unten. Zum nächsten Element gewechselt wird wie in den eingebauten EXCEL-Dialogen durch Drücken der Tabulator-Taste. Umschalt+Tabulator wechselt zum vorhergehenden Element.

EXCEL legt die Tabulator-Reihenfolge der Elemente selbst fest; es ist die Reihenfolge, in der Sie Elemente in das Dialogfeld stellen. Das entspricht wahrscheinlich nur selten dem, was Sie wünschen. Über das Menü „Extras“ und den Befehl „Aktivierfolge“ können Sie nachträglich die Reihenfolge verändern.

© by Josef Broukal, Vienna, Austria

Code für das Füllen der Elemente vor dem Anzeigen des Dialogfeldes schreiben

Alle Elemente eines Dialogfeldes behalten nach dessen Schliessung die eingegebenen Werte. Wird das Dialogfeld nochmals ausgerufen, zeigt es diese. Das wird meistens nicht passen. Daher müssen die Elemente—noch bevor das Dialogfeld angezeigt wird—mittels Code die gewünschten Werte erhalten.

- Eingabefeld mit Text belegen:

```
ThisWorkbook.Dialogsheets(„MeinDialog“).Editboxes(„DlgM_EB_Name“). _
  Text=“Müller“
```

- Listenfeld mit Inhalt von Zellen einer Tabelle füllen:

```
ThisWorkbook.Dialogsheets(„MeinDialog“).Listboxes(„DlgM_LB_Abteilungen“). _
  List=ThisWorkbook.Sheets(„Abteilungen“).[A1].CurrentRegion
```

- Erstes Element des Listenfeldes auswählen und aktiv machen:

```
ThisWorkbook.Dialogsheets(„MeinDialog“).Listboxes(„DlgM_LB_Abteilungen“). _
  Listindex=1
```

- Optionsschaltfläche „einschalten“:

```
ThisWorkbook.Dialogsheets(„MeinDialog“). _
  OptionButtons(„DlgM_OB_Gruppe“).Value=xlOn
```

Die Werte mehrerer Elemente zugleich festlegen Das geht ganz einfach! Hier kommt uns nämlich die Tatsache entgegen, daß jedes Element ein Mitglied einer Element-Auflistung ist. Daher können wir zum Beispiel:

- Text in allen Eingabefeldern löschen, indem wir ihnen eine leere Zeichenkette zuweisen:

```
ThisWorkbook.Dialogsheets(„MeinDialog“).Editboxes(„DlgM_EB_Name“). _
  Text=““
```

- Alle Kontrollkästchen ausschalten:

```
ThisWorkbook.Dialogsheets(„MeinDialog“).Checkboxes.Value=xlOff
```

- In allen Listenfeldern den ersten Eintrag auswählen:

```
ThisWorkbook.Dialogsheets(„MeinDialog“).Listboxes.Listindex=1
```

Auch während der Dialog angezeigt wird, können seine Elemente beeinflußt werden. Ansprache wie oben, allerdings genügt jetzt für die Angabe, um welchen Dialog es sich handelt, „ActiveDialog“:

```
ActiveDialog.Editboxes(„DlgM_EB_Name“).Text=“Müller“
```

Code für die laufende Kontrolle des Dialoges schreiben

Alle aktivierbaren Dialogelemente reagieren, wenn sie verändert werden. Damit läßt sich schon während der Eingabe in ein Dialogfeld ein wenig kontrollieren, ob die Eingabe sinnvoll ist. Wird ein Dialogelemente verändert, prüft EXCEL, ob es eine Prozedur gibt, die mit dem Element „verknüpft“ wurde. Ist das der Fall, dann wird diese Prozedur ausgeführt. Im VBA-Handbuch werden diese Prozeduren „Ereignis-Prozeduren“ genannt. Hier sind die „Ereignisse“ für die wichtigsten Dialogelemente:

Element	Ereignis	EXCEL-Standard-Bezeichnung für Ereignis

© by Josef Broukal, Vienna, Austria

Bearbeitungsfeld (Editbox)	Eingeben oder Löschen eines Zeichens	BeiÄnderung
Schaltknopf (Button)	Drücken des Schaltknopfes mit der Maus, Enter-Taste, wenn Knopf ausgewählt ist	BeiKlick
Optionsschaltfläche (OptionButton)	Ein- oder Ausschalten des Knopfes mit Maus oder Tastatur	BeiKlick
Listenfelder	Auswählen eines Listeneintrages mit Maus oder Tastatur	BeiÄnderung
Kontrollkästchen	Ein- oder Ausschalten des Kästchens mit Maus oder Tastatur	BeiKlick
Bildlaufleiste	Anklicken der Leiste	BeiÄnderung
Drehfeld	Anklicken eines der Dreihfeld-Pfeile	BeiÄnderung
Dialogfeldrahmen	Starten des Dialoges mittels VBA-Programm oder Drücken des Schaltknopfes „Dialog ausführen“ in der Symbolleiste „Dialog“	BeiAnzeigen

Ereignis-Prozedur durch EXCEL anlegen lassen Wenn ein Dialog-Element ausgewählt ist (also einen grauen Rahmen hat), können Sie EXCEL veranlassen, den Kopf und die Schlußzeile der für das Element zu schaffenden Ereignis-Prozedur selbsttätig anzulegen. Klicken Sie dazu in der Symbolleiste „Dialog“ auf das Symbol „Code bearbeiten“.

Wenn Sie das in einem bestimmten Dialogfeld das erste Mal machen, legt EXCEL in der Arbeitsmappe, in der sich das Dialogblatt befindet, ein neues VBA-Modul an. Weitere Prozedur-Köpfe stellt EXCEL dann in dieses Modul hinein.

Zugleich mit dem Anlegen des Prozedur-Kopfes legt EXCEL fest, daß diese Prozedur bei einem auslösenden Element-Ereignis abgearbeitet werden soll.

Ereignis-Prozedur selbst anlegen Sie können die Prozedur auch komplett selbst und in einem VBA-Modul Ihrer Wahl schreiben. Markieren Sie dann das Element. Rufen Sie im Menü „Extras“ den Befehl „Zuweisen...“ auf. Dort finden Sie in der Liste aller geladenen VBA-Prozeduren die von Ihnen geschriebene. Klicken Sie mit der Maus auf den Namen der Prozedur.

Element eine andere Ereignis-Prozedur zuweisen Markieren Sie zunächst das Element. Rufen Sie dann im Menü „Extras“ den Befehl „Zuweisen...“ auf. Dort finden Sie eine Liste aller geladenen VBA-Prozeduren. Klicken Sie mit der Maus auf den Namen der Prozedur, die bei einem „Ereignis“ des Elements ausgeführt werden soll.

Verbindung zwischen Element und Ereignis-Prozedur trennen Markieren Sie zunächst das Element. Rufen Sie dann im Menü „Extras“ den Befehl „Zuweisen...“ auf. Löschen Sie den Text im Eingabefeld „Makroname/Prozedur“.

Zur Ereignis-Prozedur springen Markieren Sie das Element. Drücken Sie dann in der Symbolleiste „Dialog“ auf das Symbol „Code bearbeiten“.

Beispiele

Eingabefeld darf nur Zahl enthalten Wir können zwar anordnen, daß ein Eingabefeld nur eine Zahl enthalten darf (über das Menü „Format“ und den Befehl „Objekt...“), aber diese Prüfung wird erst durchgeführt, wenn wir das Dialogfeld schliessen. Wollen wir schon während der Eingabe prüfen, verwenden wir die VBA-Funktion „IsNumeric“. Sie ergibt „Wahr“, wenn der Text im Eingabefeld nur Ziffern, ein Komma und Tausenderpunkte enthält. Die Ereignis-Prozedur „PruefeObZahl“ stellt nach

© by Josef Broukal, Vienna, Austria

jedem Tastendruck fest, ob das eingebene Zeichen „numeric“ ist. Wenn nicht, erfolgt eine Warnung und das letzte, offensichtlich falsche Zeichen wird aus dem Eingabefeld-Text entfernt.

Den Code dieses Beispiels finden Sie im Verzeichnis „Dialog“ in der Datei „ZAHL.XLS“.

```
Sub PruefeObZahl()

Dim sZuPruefendesEingabefeld As EditBox

Set sZuPruefendesEingabefeld = _
  ActiveDialog. Editboxes(Application.Caller)
If Not IsNumeric(sZuPruefendesEingabefeld.Text) Then
  Beep
  MsgBox "Dies ist keine Zahl !", vbCritical, "Fehler:"
  sZuPruefendesEingabefeld.Text = Left(sZuPruefendesEingabefeld.Text, _
    Len(sZuPruefendesEingabefeld.Text) - 1)
  SendKeys "{end}"
End If

End Sub
```

Hinweis „Isnumeric“ ist ein bißchen dumm. Man kann ihm an beliebiger Stelle Tausender-Punkte unterschieben, mehr als ein Dezimalkomma läßt es nicht zu.

Was zeigt dieses Beispiel noch? Zunächst einmal ist diese Prozedur universell verwendbar. Sie stellt nämlich nicht auf ein konkretes Eingabefeld ab, sondern sie erfragt dessen Namen, während sie läuft:

```
Dim sZuPruefendesEingabefeld As EditBox

Set sZuPruefendesEingabefeld = _
  ActiveDialog.Editboxes(Application.Caller)
```

Die Eigenschaft „Caller“ des „Application“-Objektes gibt den Namen des Dialog-Elementes zurück, durch dessen Veränderung die Prozedur „PruefeObZahl“ aufgerufen wurde.

Da EXCEL darauf besteht, nach der Eingabe eines Textes in ein Eingabefeld durch VBA-Code den Text komplett zu markieren, sodaß jeder Tastendruck ihn ganz verschwinden lassen würde, ordnen wir in der drittletzten Zeile an:

```
SendKeys "{end}"
```

Das entspricht dem Drücken der „Ende“-Taste. Die Markierung wird aufgehoben, der Cursor wird an das Ende der verbliebenen Zeichen gesetzt.

Abhängige Felder automatisch ausfüllen Angenommen, der Benützer eines VBA-Programmes kann in einem Dialogfeld aus einer Mitarbeiterliste auswählen. In einer Tabelle sind zu jedem Mitarbeiter auch Abteilung, Zimmernummer und Telefonklappe aufgeführt. Daher liegt es nahe, nach Auswahl des Mitarbeiters die anderen Eingabefelder automatisch ausfüllen zu lassen.

© by Josef Broukal, Vienna, Austria

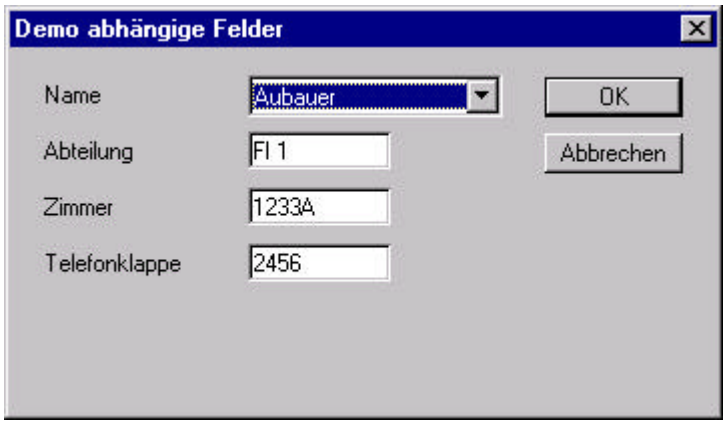


Abbildung 38: Wird aus dem Listenfeld „Name“ ein Name ausgewählt, stellt eine „Ereignis-Prozedur“ die zum Mitarbeiter gehörenden Daten für Abteilung, Zimmer und Telefonklappe in die Eingabefelder.

Hier ist der Code:

```
'Objekt-Variable für Dialog1 deklarieren
Private Dialog1_DD_Name As DropDown
Private Dialog1_EB_Abteilung As EditBox
Private Dialog1_EB_Telefonklappe As EditBox
Private Dialog1_EB_Zimmer As EditBox

Private rMitarbeiterDaten As Range

'Objekt-Variablen die Dialogelemente zuweisen
Sub SetzeObjektVariableFürDialog1()

With DialogSheets("Dialog1").DrawingObjects
  Set Dialog1_DD_Name = .Item("Dialog1_DD_Name")
  Set Dialog1_EB_Abteilung = .Item("Dialog1_EB_Abteilung")
  Set Dialog1_EB_Telefonklappe = .Item("Dialog1_EB_Telefonklappe")
  Set Dialog1_EB_Zimmer = .Item("Dialog1_EB_Zimmer")
End With

End Sub

'Dialogfeld-Elemente auf Anzeige vorbereiten
Sub Dialog1_BeiAnzeigen()

  Call SetzeObjektVariableFürDialog1

  Dialog1_DD_Name.OnAction = "Dialog1_DD_Name_EreignisProzedur"

  'Tabellenbereich mit Mitarbeiter-Daten Objekt-Variabler zuweisen
  Set rMitarbeiterDaten = _
    ThisWorkbook.Sheets("Tabelle1").Cells(1, 1).CurrentRegion
  Set rMitarbeiterDaten = _
    rMitarbeiterDaten.Offset(1).Resize(rMitarbeiterDaten.Rows.Count - 1)

  Dialog1_DD_Name.List = rMitarbeiterDaten.Columns(1)
  Dialog1_DD_Name.ListIndex = 1

  Call Dialog1_DD_Name_EreignisProzedur

End Sub
```

© by Josef Broukal, Vienna, Austria

```
'Abhängige Bearbeitungsfelder nach Auswahl eines Namens
'füllen
Sub Dialog1_DD_Name_EreignisProzedur()

Dialog1_EB_Abteilung.Text = _
  rMitarbeiterDaten.Cells(Dialog1_DD_Name.ListIndex, 2).Value
Dialog1_EB_Zimmer.Text = _
  rMitarbeiterDaten.Cells(Dialog1_DD_Name.ListIndex, 3).Value
Dialog1_EB_Telefonklappe.Text = _
  rMitarbeiterDaten.Cells(Dialog1_DD_Name.ListIndex, 4).Value

End Sub
```

Erklärung Wie haben wir den Abteilungs-Namen gefunden? Wir fragen zunächst die „Listindex"-Eigenschaft des Listenfeldes ab. Sie gibt uns eine Zahl zurück, die laufende Nummer des ausgewählten Eintrages im Listenfeld. Angenommen, wir haben „Mück" gewählt, dann ist das eine Zwei, denn Mück ist der zweite Eintrag. Genauso ist „Mück" aber der zweite Eintrag im Tabellenbereich „rMitarbeiterDaten", aus dem wir die Liste gefüllt haben. Daher finden wir den Abteilungsnamen in jener Zelle von „rMitarbeiterDaten", die in der zweiten Zeile und in der zweiten Spalte steht. Obwohl „rMitarbeiterDaten" nur eine Spalte breit ist, können wir mit der „Cells"-Methode über diese Grenze hinausgreifen. In Spalte Drei finden wir dann die Zimmer-Nummer, in Spalte Vier die Telefonklappe. **Wichtig:** Spalte Eins ist nicht notwendig Spalte A. Es ist, wie die folgende Abbildung zeigt, die erste Spalte von „rMitarbeiterDaten"!

	A	B	C	D	E
1					
2					
3		*Spalte 1*	*Spalte 2*	*Spalte 3*	*Spalte 4*
4		**Name**	**Abteilung**	**Zimmer**	**Klappe**
5	*Zeile 1*	Aubauer	Fl 1	1233A	2456
6	*Zeile 2*	Mück	Fl 2	4233	2622
7	*Zeile 3*	Broukal	Fl 1	1265A	2345
8					
9		*rMitarbeiterDaten*			
10					

Abbildung 39: Der Tabellenbereich „rMitarbeiterDaten" ist zwar nur eine Spalte breit, aber wir können dennoch feststellen, welchen Wert eine Zelle in (vom Bereich weg gesehen) Spalte 2, Spalte 3 usw hat. Die „Cells"-Methode macht es möglich.

Beispiele zu diesem Kapitel finden Sie im Verzeichnis „Dialog" in der Datei „ABHAENGIG.XLS".

Dialogfeld-Elemente im Überblick

DIALOGFRAME (DIALOGRAHMEN)

Der Dialogfeldrahmen stellt den Hintergrund für benutzerdefinierte Dialogfelder dar. Er weist keine Formatierungseigenschaften auf, sondern lediglich eine Position, eine Größe und einen Titel. Pro Dialogblatt gibt es nur einen Dialogfeldrahmen, der weder gelöscht, noch vor ein anderes Grafikobjekt geschoben werden kann. In Tabellenblättern gibt es keine Dialogfeldrahmen.

© by Josef Broukal, Vienna, Austria

Wichtige Eigenschaften des Dialograhmens

Eigenschaft	Werte	bewirkt
Text	beliebige Zeichen	Setzt/ändert Titel des Dialogfeldes
Left, Top, Height, Width	Zahl	Setzt/ändert Position und Größe des Dialogfeldes. Kann auch verändert werden, während der Dialog angezeigt wird

DIE DRAWINGOBJECTS-AUFLISTUNG

Alle Dialogelemente auf einem Dialogblatt sind Teil der „DrawingObjects“-Auflistung. Dieses Container-Objekt ist wichtig, wenn es gilt, alle Elemente eines Dialoges zu untersuchen. Die Datei „DlgVars.XLS“ wäre ohne dieses Objekt kaum denkbar: Sie schreibt automatisch den Code für ein komplettes Dialogfeld in ein Modulblatt. Dabei untersucht sie Element für Element in dessen Eigenschaft als Mitglied der „DrawingObjects“.

Das Programm „DlgVars.XLS“ finden Sie im Verzeichnis „Dialog“

Gemeinsame Eigenschaften aller Dialog-Elemente

Eigenschaft	Werte	bewirkt
Visible	True, False	Element ist sichtbar/nicht sichtbar
Enabled	True, False	Element kann mit Tastatur oder Maus geändert werden/nicht geändert werden
Top, Left	Zahl	Position im Dialograhmen **Wichtig:** Kann nur gesetzt werden, wenn der Dialog nicht angezeigt wird

TEXTBOX (TEXTFELD)

Statischer Text zur Erklärung der Dialogelemente für Ein- und Ausgabe. Der Inhalt des Feldes kann über die Eigenschaft „Text“ sowohl vor dem Anzeigen des Dialogs als auch zur Laufzeit des Dialogs verändert werden:

```
ActiveDialog.[textfeldname].Text=„Soeben verändert“
```

EDITBOX (BEARBEITUNGSFELD)

Nimmt Text durch Tastatureingabe auf oder zeigt Text an, der über ein VBA–Programm in das Feld gestellt wurde. Der Inhalt des Feldes kann über die Eigenschaft „Value“ zur Laufzeit des Dialogs verändert werden:

```
ActiveDialog.[bearbeitungsfeldname].Value=„Soeben verändert“
```

Gleichermaßen kann der Inhalt ausgelesen werden:

```
Variable= ActiveDialog.[bearbeitungsfeldname].Value
```

Der Text im Bearbeitungsfeld wird durch Zuweisung einer leeren Zeichenkette gelöscht:

© by Josef Broukal, Vienna, Austria

```
ActiveDialog.[bearbeitungsfeldname].Value =“”
```

Das Bearbeitungsfeld kann auch nur zur Anzeige eines Textes verwendet werden. Dies wird über die „Wählbar“–Eigenschaft gesteuert:

```
ActiveDialog.[bearbeitungsfeldname].Enabled=False
```

Schließlich kann ein Bearbeitungsfeld auch unsichtbar gemacht werden.

```
ActiveDialog.[bearbeitungsfeldname].Visible=False
```

Kontrollmöglichkeiten während des Programm-Laufs Während Text in einer Editbox eingegeben oder bearbeitet wird, verzweigt VBA nach jedem Tastendruck in die zugehörige Ereignis-Prozedur. So läßt sich zum Beispiel prüfen, ob der eingegebene Text eine Zahl ergibt (siehe oben die Prozedur „PruefeObZahl“).

GROUPBOX (GRUPPENFELD)

Rahmen zur optischen Gestaltung eines größeren Dialogfeldes und zur Steuerung einander gegenseitig ausschließender Optionen. Für alle mit ihrer linken oberen Ecke im Groupbox–Rahmen stehenden Optionsfelder gilt, daß nur eines von ihnen ausgewählt sein kann. Die (von oben nach unten) fortlaufende Nummer der im Gruppenfeld ausgewählten Option können Sie in einer Tabellenzelle ablegen und erfragen. Es gibt aber in EXCEL 7.0 keine Möglichkeit, mit einem VBA-Befehl die Nummer zu erfragen. Stattdessen muß jedes Optionsfeld einzeln geprüft werden.

BUTTON (SCHALTFLÄCHE)

Löst Vorgänge aus. Die Schaltflächen „OK“ und „Abbrechen“sind standardmäßig vorgegeben, weitere Schaltflächen werden beim Zeichnen des Dialogfeldes dazugefügt. Jeder Schaltfläche kann (über das Menü „Format“ und den Befehl „Objekt...“) eine oder mehrere der vier Funktionsweisen „Standard“, “Abbrechen“, „Schließen“ und „Hilfe“ zugeordnet werden:

Standard	ist ausgewählt (stark umrandet); wird die Enter–Taste gedrückt, verhält sich der Dialog so, als wäre die Standard–Schaltfläche angeklickt worden
Abbrechen	Dialog gilt als „Abgebrochen“, gibt an den aufrufenden Code den Wert „False“ zurück
Schließen	Dialog wird beendet, Dialogfeld wird geschlossen. Gibt an den aufrufenden Code „True“ zurück
Hilfe	Schaltfläche erhält den Text „Hilfe“

Diese Funktionsweisen können Schaltflächen auch durch VBA-Code zugewiesen werden oder durch Code erfragt werden:

Button-Objekt.**DefaultButton** („True“ oder „False“)
Button-Objekt.**CancelButton** („True“ oder „False“)
Button-Obkekt.**DismissButton** („True“ oder „False“)
Button-Objekt.**HelpButton** („True“ oder „False“)

Diese Eigenschaften eines Buttons können nebeneinander festgelegt werden. Einem Standard-Button kann zum Beispiel die (immer vorgegebene) Eigenschaft, bei Anklicken das Dialogfeld zu schließen, entzogen werden:

```
ActiveDialog.Buttons(„Dlg_BT_OK“).DissmissButton=False
```

Button-Aufschrift ändern Die Beschriftung jedes Buttons ist durch seine „Caption“-Eigenschaft festgelegt. Diese kann erfragt und geändert werden, bevor oder während der Dialog angezeigt wird:

© by Josef Broukal, Vienna, Austria

```
If ActiveDialog.Buttons("Dlg_BT_OK").Caption = "Guten Abend" Then
  ActiveDialog.Buttons("Dlg_BT_OK").Caption = "Guten Morgen"
End If
```

Hinweis Egal, wie hoch Sie einen Schaltknopf machen, Text stellt er immer nur in einer Zeile dar.

CHECKBOX (KONTROLLKÄSTCHEN)

Kennt drei Werte: Angekreuzt, nicht angekreuzt, teilweise angekreuzt. Der Zustand kann mit Hilfe der „Value"–Eigenschaft und dreier Konstanten erfragt oder gesetzt werden:

```
If ActiveDialog.[kontrollkästchen].Value= xlOn Then
  ActiveDialog.[kontrollkästchen].Value = xlOff
ElseIf ActiveDialog.[kontrollkästchen].Value = xlOff Then
 ActiveDialog.[kontrollkästchen].Value = xlMixed
End If
```

OPTIONBOX (OPTIONSFELD)

Kann ausgewählt oder nicht ausgewählt sein. Liegen mehrere Optionsfelder mit ihrer linken oberen Ecke in einem Gruppenfeld–Rahmen, sorgt EXCEL dafür, daß jeweils nur eines von ihnen ausgewählt werden kann.

```
If ActiveDialog.[Optionsfeld].Value= xlOn Then
  ActiveDialog.[ Optionsfeld].Value = xlOff
ElseIf ActiveDialog.[ Optionsfeld].Value = xlOff
  …
End If
```

LISTBOX (LISTENFELD)

Bildet die Einträge in einem zusammenhängenden Bereich einer Tabelle ab. Es können, je nach Einstellung des Listenfeldes über FORMAT, OBJEKT

- ein,
- mehrere zusammenhängende,
- mehrere nicht zusammenhängende Einträge

ausgewählt werden. Der Tabellenbereich, der die Liste enthält, kann in diesem Dialog im Bearbeitungsfeld „Listenbereich" angegeben werden.

Ein „einfaches" Listenfeld gibt die Zahl des Elementes in der Liste zurück, welches ausgewählt worden ist. Die Zahl wird in jener Zelle, deren Bezug im Dialog FORMAT, OBJEKT ins Feld „Ausgabeverknüpfung" eingetragen worden ist, hinterlegt. Sie kann auch durch Abfragen der Eigenschaft „Listindex" erfragt werden. Die beiden eine Mehrfachauswahl zulassenden Listenfeldtypen liefern Datenfelder, die die Position der ausgewählten Listeneinträge enthalten. Leider führt die in der Online–Hilfe für EXCEL 5.0 angegebene Syntax für die Mehrfachauswahl zu einem Laufzeitfehler, sodaß dafür keine Beispiele gegeben werden können. In EXCEL für Windows95 ist Mehrfach-Auswahl wie in der Online-Hilfe beschrieben möglich.

Listenfelder werden über folgende Eigenschaften gesteuert:

© by Josef Broukal, Vienna, Austria

Eigenschaft	verlangt	bewirkt
List	Datenfeld, Bereichsobjekt	Liste wird mit den Elementen des Datenfeldes gefüllt
List(Zahl)	Zahl zw. 1 und Anzahl der Elemente im Listenfeld	Beim Setzen: legt den Wert des Listenfeld-Eintrages „Zahl“ fest Beim Lesen: Gibt den Wert des Listenelementes an der Stelle von „Zahl“ aus
ListIndex(Zahl)	Zahl	Beim Setzen: Legt das aktive Element der Liste fest Beim Lesen: gibt Position des aktiven Elementes in der Liste zurück
ListCount		Gibt Anzahl der Elemente im Listenfeld zurück
ListFillRange	Verweis auf einen Tabellenbereich in Textform	Beim Setzen: Liste wird mit dem Wert der Zellen des Tabellenbereiches gefüllt Beim Lesen: gibt den Tabellenbereich als Text zurück, aus dem die Liste gefüllt wurde

Listenfelder werden über folgende Methoden verändert:

Methode	bewirkt
AddItem	Liste erhält ein zusätzliches Element
RemoveItem	Löscht ein Element aus der Liste
RemoveAllItems	Löscht die ganze Liste

„List“ oder „ListFillRange“, das ist hier die Frage...

„List“ füllt die Liste aus den Einträgen eines Datenfeldes, „ListFillRange“ aus den Zellen eines Tabellenbereiches. Verwenden Sie dennoch auch für Tabellen-Listen „List“, den „ListFillRange“ gibt sowohl beim Löschen als auch beim Hinzufügen von Listeneinträgen den Geist auf. Und EXCEL ist gescheit genug, die Zellen eines Tabellenbereiches automatisch in ein Datenfeld umzuwandeln, sobald „List“ mit Hilfe einer Objekt-Variablen der Tabellenbereich gezeigt wird.

Die „List“-Eigenschaft

Weist einer Liste die Elemente eines Datenfeldes oder eines als Objekt-Referenz vorliegenden Zellbereiches zu.

Syntax 1 - ganze Liste

Objekt.List

Beispiel

```
Set rDieListe = ThisWorkbook.Sheets("Tabelle1"). _
  Cells(1, 1).CurrentRegion
Dlg1_Listbox_1.List = rDieListe
```

Listenbereich der Tabelle wird mit einem EXCEL-Namen angesprochen Angenommen, Sie haben in einer Tabelle einen Bereich von Zellen mit einem Namen versehen (in unserem Fall „TabellenZellenFuerListe“), dann können Sie mit Hilfe der Methode „Evaluate“ diese Zelladresse in eine Objekt-Referenz umwandeln und die Referenz dann der „List“-Eigenschaft zuweisen:

© by Josef Broukal, Vienna, Austria

```
ThisWorkbook.DialogSheets("Dialog1"). _
  ListBoxes("Dialog1_LB_DieListe").List =
Evaluate("TabellenZellenFuerListe")
```

Listenbereich der Tabelle wird mit Zellreferenz angesprochen Auch hier hilft wieder die Methode „Evaluate“ weiter:

```
Sheets("Dialog1").ListBoxes("Dialog1_LB_DieListe").List = _
  Evaluate("[ListeNeu.xls]Tabelle1!C1:C12")
```

Syntax 2 - einzelne Listenelemente

Objekt.List(Index)

Wert eines Listen-Elementes erfragen Dazu weisen wir das Ergebnis von „List(Index)“ einer Variablen zu oder verwenden es in einer Anweisung:

```
' Wert von Listen-Element 6 erfragen
vListenwert=Dlg1_DasListenfeld.List(6)
' Wert des aktiven, gerade ausgewählten Listen-Elementes erfragen
vListenwert=Dlg1_DasListenfeld.List(Dlg1_DasListenfeld.ListIndex)

' Inhalt von Element 6 prüfen
If Dlg1_DasListenfeld.List(6)= "Guten Morgen" Then ...
```

Wert eines Listen-Elementes setzen Dazu weisen wir dem Listen-Element den Wert zu:

```
sAntwort = InputBox("Bitte geben Sie den neuen Text " & _
    "des Listeneintrags an:")
  If sAntwort = "" Then Exit Sub
  Dialog1_LB_DieListe.List(Dialog1_LB_DieListe.ListIndex) = sAntwort
```

Die „ListIndex“-Eigenschaft

Syntax

Objekt.ListIndex

Gibt die fortlaufende Nummer des aktiven Listen-Elementes wieder. Diese Eigenschaft legt fest oder informiert darüber, welches Element der Liste „aktiv“ ist, also ausgewählt erscheint oder ausgewählt wurde. „ListIndex“ ist immer eine Zahl.

Beispiele

Element 4 der Liste auswählen (erscheint mit Balken unterlegt):

```
[Listenfeld-Referenz].ListIndex=4
```

Fortlaufende Nummer des Elements innerhalb der Liste anzeigen, das ausgewählt wurde:

```
MsgBox [Listenfeld-Referenz].ListIndex
```

Inhalt des ausgewählten Listenelementes erfragen

Wenn der Listenbereich als Text vorliegt und “Listenindex“ den Wert 2 hat.

© by Josef Broukal, Vienna, Austria

```
MsgBox Application.Index([[MAPPE1]Tabelle1!A1:A5], 2)
```

Beachten Sie die beiden eckigen Klammerpaare!

Die „ListCount“-Eigenschaft

Gibt als Zahl die Anzahl der Listen-Elemente wieder. Nützlich, wenn es darum geht, alle Elemente einer Liste in eine Tabelle zu schreiben:

```
For iZaehler = 1 To Dialog1_LB_DieListe.ListCount
      rListenBereich.Cells(iZaehler, 1).Value =
Dialog1_LB_DieListe.List(iZaehler)
Next
```

ListFillRange (Listenbereich)

Die Adresse des Tabellenbereiches, der die Einträge der Liste enthält, in Text-Form. **Achtung:** die Adresse **muß** in Textform angegeben werden; mit Objektvariablen, die eine Referenz auf den Tabellenbereich enthalten, kann „Listenbereich“ nicht gefüttert werden!

Beispiel:

```
[Listenfeld-Referenz].ListFillRange=“[MAPPE5.XLS]Tabelle3!A2:A14“
```

nicht aber:

```
Set Namenszellen=Workbooks(“Mappe5.xls“)._
  Sheets(“Tabelle3“).Range(“A2:A14“)

[Listenfeld-Referenz]. ListFillRange =Namenszellen
```

So geht es aber mit einer Objektvariablen:

```
[Listenfeld-Referenz]. ListFillRange=Namenszellen.Address (external:=True)
```

Die Address-Methode macht aus einer Objekt-Referenz wieder einen Referenz-Text. Das benannte Argument “external“ stellt den vollständigen Text der Referenz bereit: Mappe, Blatt und Zellen.

DROPDOWN–LISTENFELD

Idente Funktionalität mit dem Listenfeld, jedoch muß die Liste durch Klicken auf die Schaltfläche am rechten Rand oder durch Drücken der Tasten „Alt“+“Pfeil-ab“ erst aufgeklappt werden.

KOMBINATIONSFELD LISTE–TEXT

Besteht aus einem Bearbeitungsfeld und einem Listenfeld. Die beiden sind unabhängige Dialogfeld–Elemente, allerdings mit folgender Funktionalität verbunden:

- Auswahl eines Eintrages im Listenfeld stellt den Text des Eintrages in das Bearbeitungsfeld; der Index des Eintrages in der Listen–Matrix wird in die für die Ausgabeverknüpfung bestimmte Zelle gestellt.

Hinweis Ein Eintrag in das Bearbeitungsfeld wird nicht im Listenfeld gesucht. Selbst wenn er dort vorhanden ist, enthält die Ausgabeverknüpfungs–Zelle den Wert 0. Sie müssen also mit VBA-Code die ganze Liste durchsuchen. Wird der Text des Bearbeitungsfeldes nicht gefunden, müssen Sie ihn mit VBA-Code in die Liste einfügen.

© by Josef Broukal, Vienna, Austria

Code für die praktische Arbeit mit einem Kombinationsfeld Liste-Text finden Sie in den Datei „KombiListe.XLS“ und „KombiFeld alt.XLS“ im Verzeichnis „Dialog“

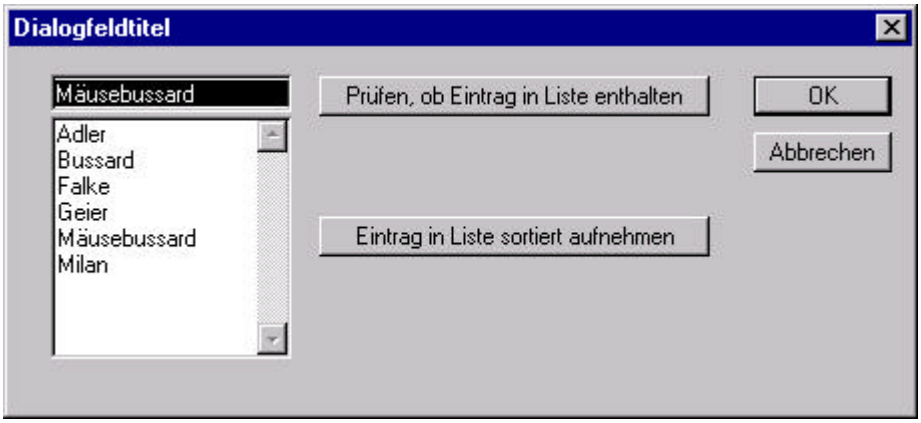


Abbildung 40: „KombiListe.XLS“ enthält Beispielscode für die Arbeit mit einem Kombinationsfeld Liste-Text. Nach jedem Eintrag kann durch Drücken auf den oberen Schaltknopf („Prüfen...“) festgestellt werden, ob der Eintrag schon in der Liste enthalten ist. Wenn ja, erfolgt eine Warnung. Der Schaltknopf „Eintrag in Liste sortiert aufnehmen“ vergrößert den Tabellenbereich, aus dem das Listenfeld gespeist wird, um eine Zeile, schreibt in diese Zeile den neuen Eintrag, sortiert den Tabellenbereich und weist ihn als Datenquelle wieder dem Listenfeld zu.

DROPDOWN LISTE–TEXT

Mit Kombinationsfeld Liste–Text idente Funktionalität, doch muß die Liste durch Klicken auf die Schaltfläche am rechten Rand oder durch Drücken der Tasten „Alt“+“Pfeil-ab“ erst aufgeklappt werden.

BILDLAUFLEISTE

Funktionalität wie in der in EXCEL eingebauten Datenmaske. Die Bildlaufleiste verwaltet eine ganze Zahl zwischen den beiden (einstellbaren) Eckwerten „Min“ und „Max“. Diese Zahl kann über die Eigenschaft „Value“ ausgelesen und auch eingestellt werden. Jedes Klicken auf die Pfeile am oberen bzw. unteren Ende der Bildlaufleiste erhöht bzw. vermindert den aktuellen Wert der Bildlaufleiste. Der kleinste Wert („Min“-Eigenschaft) ist 0, der größte („Max“-Eigenschaft) ist 30000. Zusätzlich kann ein Wert für „LargeChange“ eingestellt werden. Diese „Große Bewegung“ wird ausgelöst, wenn Sie mit der Maus nicht auf den „Fahrstuhl“ klicken, sondern auf die Flächen darüber und darunter.

DREHFELD

In der Funktionalität mit der Bildlaufleiste ident bis auf das Fehlen von „LargeChange“. Drehfelder werden wahrscheinlich immer im Zusammenhang mit Bearbeitungsfeldern eingesetzt. Der Gleichklang zwischen Drehfeld und Bearbeitungsfeld ist eine verzwickte Aufgabe:

- Wird als Eingabe eine Zahl verlangt, ist die Prüfung des zugeordneten Bearbeitungsfeldes eine kompexe Sache. Am Beginn der Schwierigkeiten steht schon, daß EXCEL-Dialogfelder keine Ahnung davon haben, ob sie aktiviert oder verlassen wurden. Das einzige, was sie registrieren, ist, daß ein Tastendruck erfolgt. Daher muß man nach jedem Tastendruck prüfen: a) ist der Maximalwert überschritten?, b) wenn die Anzahl von Stellen des Minimalwertes erreicht, aber der Maximalwert nicht überschritten ist, ob beim Anfügen einer weiteren Ziffer ins Bearbeitungsfeld der Maximalwert überschritten würde (man fügt dafür eine Null an den augenblicklichen Wert des Bearbeitungsfeldes an).

© by Josef Broukal, Vienna, Austria

- Die Obergrenze des Drehfeldwertes von 30.000 ist zu klein, um ein Datum in den neunziger Jahren darstellen zu können. Soll ein Drehfeld also ein Bearbeitungsfeld steuern, das ein Datum enthalten soll, muß jede Bewegung als „gehe Eins weiter“ bzw. „gehe Eins zurück“ interpretiert werden. Dann muß aus dem zugeordneten Bearbeitungsfeld das Datum entnommen und entsprechend der Bewegung verändert werden. Der Wert des Drehfeldes (minus Eins bzw. plus Eins) ist nach erfolgter Anpassung des Eingabefeldes wieder auf 0 zu stellen.

Beispiele für das Arbeiten mit Drehfeldern finden Sie im Verzeichnis „Dialog“ in der Datei „Drehfeld.XLS“. Der Modul „Mod DialogDrehfeld“ enthält den Code für das Prüfen einer numerischen Eingabe, die zwischen den Grenzen eines Drehfeldes liegen muß. „Mod HalbstundenDrehfeld“ zeigt in seinem Bearbeitungsfeld die Uhrzeit; diese kann mit dem Drehfeld halbstundenweise verändert werden.

© by Josef Broukal, Vienna, Austria

Host-Daten einlesen

Das Übernehmen von Daten von Servern und Hosts gehört zu den Anwendungen, die besonders gerne mit Code gelöst werden. Die dabei gegebenen Einstellmöglichkeiten sind sehr komplex—außer, es wird schon im Server eine EXCEL–Datei eingerichtet. Das dürfte aber selten der Fall sein; meistens haben wir es mit ASCII Text–Dateien zu tun.

Ein ausprogrammiertes Beispiel finden Sie im Verzeichnis „Lesehost“ in der Datei „LESEHOST.XLS“.

EXCEL sieht nur Dateien mit der Endung TXT als Textdateien an. Dateien mit den Endungen CVS und PRN werden weder vom interaktiven Text-Assistenten noch von den entsprechenden VBA-Befehlen akzeptiert.

Hinweis “CSV“ steht zwar für „comma–separated values“, aber unter Windows ist unter „comma“ das in der Systemsteuerung eingestellte Listentrennzeichen zu verstehen—im deutschen Sprachgebiet also der Strichpunkt, im Englischen wirklich das Komma.

VBA hält für das automatischer Einlesen von Textdateien die Methode „OpenText“ bereit. Sie ist komplex, dafür aber enorm leistungsfähig.

Sonderfälle

Nicht immer liegen Hostdaten in einer Form vor, die für die nahtlose Übernahme in EXCEL geeignet ist. Zwei dieser Sonderfälle möchte ich kurz behandeln.

Datum als Zeichenkette ohne Trennzeichen

Oft liegt in einer Übergabe–Datei das Datum als Zeichenkette in der Form TTMMJJ vor. EXCEL kann nicht erkennen, daß es sich dabei um ein Datum handelt. Es interpretiert die nur aus Ziffern bestehende Zeichenkette als Zahl. Zusätzlich wird die führende Null bei den Tagen 1 bis 9 nicht übernommen.

Das Ergebnis ist also eine Zahl, deren einzelne Ziffern für TTMMJJ bzw. TMMJJ stehen.

Hier hilft nur die Rückwandlung der Zahl in eine Zeichenkette. Aus dieser werden dann von hinten zunächst zwei Ziffern für das Jahr herausgeschnitten, dann zwei Ziffern für den Monat. Der Rest steht dann, egal ob es eine oder zwei Ziffern sind, für den Tag. Die drei Teil–Zeichenketten werden über die Funktion „DateSerial“ in ein Datum umgewandelt.

Ein Beispiel dafür finden Sie im Verzeichnis „Lesehost“ in der Datei „LESEHOST.XLS“ im VBA–Modul „Mod DatumUmwandeln“.

Kennziffern mit führenden Nullen

Hier ist das selbe Problem gegeben. Die Zeichenkette wird in eine Zahl umgewandelt, führende Nullen werden weggelassen. Dies kann nur dadurch unterdrückt werden, daß beim Anwenden der VBA–Methode „OpenText“ eine Feld–Info für die Spalte angegeben wird, in der die führenden Nullen erhalten bleiben sollen. Diese Feld–Info muß dem Feld den Typ „Text“ zuweisen.

© by Josef Broukal, Vienna, Austria

Ein Beispiel dafür finden Sie im Verzeichnis „Lesehost“ in der Datei „LESEHOST.XLS“ im VBA–Modul „Mod FührendeNullenBehalten“

© by Josef Broukal, Vienna, Austria

EXCEL Makros und VBA — einmal hin und zurück

Zum Aufruf eines Makros der alten EXCEL-Programmiersprache in einer Visual Basic-Prozedur wird der Befehl "Run" benutzt. Der Makroname wird als erstes Argument in Hochkommas eingeschlossen übergeben. Weitere Argumente werden, durch Strichpunkt getrennt, in Hochkommas angefügt.

Nachfolgend finden Sie einfache Beispiele, wie Sie gleichzeitig mit beiden EXCEL-Programmiersprachen arbeiten können:

VISUAL BASIC-PROZEDUR RUFT VISUAL BASIC-PROZEDUR IN DER SELBEN ARBEITSMAPPE AUF:

```
Sub RufeVBProzedur()
  Call MeineProzedur
End Sub
```

VISUAL BASIC-PROZEDUR RUFT VISUAL BASIC-PROZEDUR IN ANDERER ARBEITSMAPPE AUF:

```
Sub RufeVBProzedurInAndererArbeitsmappe()
   Run "MAK2VBA.xls!MeineProzedur"
End Sub
```

VISUAL BASIC-PROZEDUR RUFT EXCEL-BEFEHLSMAKRO AUF:

(Dem folgendem EXCEL-Befehlsmakro wird ein Argument zugewiesen. Der Wert des Arguments wird in einem Meldungsfenster ausgegeben. Das Makro liefert keinen Rückgabewert.)

ExcelBefehlsmakro
=ARGUMENT("Eingabe")
=WARNUNG(Eingabe)
=RÜCKSPRUNG()

Diese Visual Basic Prozedur ruft das Makro "ExcelBefehlsmakro" auf und übergibt das Argument "Hallo".

```
Sub AusführenExcelBefehlsmakro()
   Run "ExcelBefehlsmakro", "Hallo, ich bin ein altes EXCEL Befehlsmakro"
End Sub
```

VISUAL BASIC-PROZEDUR RUFT EXCEL-FUNKTIONSMAKRO AUF:

Der Rückgabewert des EXCEL-Makros wird zum Rückgabewert der Ausführen-Methode. Wird der Rückgabewert der Ausführen-Methode verwendet, sollten alle Argumente von Klammern umschlossen werden.

© by Josef Broukal, Vienna, Austria

ExcelMakrofunktion
=RÜCKSPRUNG(EINGABE("Geht doch sehr gut, oder...?";2))

```
Sub AusführenExcelMakrofunktion()
  Dim Antwort
  Antwort = Run("Excel4Funktion")
  MsgBox (Antwort)
End Sub
```

EXCEL-MAKRO RUFT VISUAL BASIC-PROZEDUR AUF

```
Sub VBAProzedur(Aufruf)
  MsgBox (Aufruf)
End Sub
```

(Excel-Makro übergibt das Argument "Hallo VBA" in Klammern beim Aufruf der VB-Prozedur.)

```
AusführenEinfachBefehl
=VBAProzedur("Hallo VBA")
=Rücksprung()
```

EXCEL-MAKRO RUFT VISUAL BASIC-FUNKTION AUF:

MakroRuftVBAFunktionAuf
=WARNUNG(VBAFunktion())
=RÜCKSPRUNG()

```
Function VBAFunktion()
  VBAFunktion = InputBox("Diese EingabeBox kommt aus der VBA Funktion")
End Function
```

VBA-Äquivalente der EXCEL 4.0 Makrosprache

Microsoft gibt in der EXCEL–Hilfe eine alphabetische Liste der alten Makro–Schlüsselworter und—in kurzen Beispielen—den ihnen gleichwertigen VBA–Code. Sie finden diese Liste unter

„Inhalt der Microsoft EXCEL–Hilfe“
„Referenz“
„Inhalt der Microsoft EXCEL–Makrofunktionen“
„Visual Basic-Äquivalente von Makrofunktionen und -befehle“

Diese Übersicht ist leider nicht vollständig. In vielen Fällen kann aber mit Hilfe des Makro–Rekorders VBA-Code erzeugt werden, der Ihnen einen ersten Aufschluß über die neuen Schlüsselwörter gibt.

Den Code dieser Beispiele finden Sie im Verzeichnis „MAK2VBA“ in der Datei „MAK2VBA.XLS“

In EXCEL 7.0 finden Sie die „Visual Basic-Entsprechungen für Makrofunktionen und -befehle“ am leichtesten durch Eingabe von „Makrofunktion“ im Register „Suchen“ des „Hilfethemen“-Dialogs.

© by Josef Broukal, Vienna, Austria

Bücher und andere Hilfen

Bücher

Fast jeder Computer–Verlag hat auch ein Buch im Programm, in dem EXCEL und VBA beschrieben werden. Die meisten dieser Bücher kommen aber über „erste Schritte“ kaum hinaus. Empfehlen kann ich Ihnen lediglich

Microsoft Press, **EXCEL/Visual Basic Programmer’s Guide** für Windows 95, 1995

Erich Wells, **Developing EXCEL 5 Solutions,** Microsoft Press, 1994
Eric Wells, **Developing EXCEL 95 Solutions,** Microsoft Press, 1995

Michael Kofler, **Anwendungs–Programmierung mit EXCEL 5.0**, Addison–Wesley, 1994
Michael Kofler, **Anwendungs–Programmierung mit EXCEL 7.0**, Addison–Wesley, 1996

Koflers Bücher dokumentieren auch Fehler und Ungereimtheiten in VBA, ersparen Ihnen also das verzweifelte Bemühen, Code zum Laufen zu bringen, der nicht laufen kann.

Matthew Harris, **Teach Yourself EXCEL Programming with Visual Basic for Applications in 21 Days,** SAMS Publishing, Indianapolis, USA, 1996

Viele kleine Kapitel, sehr übersichtlich, leider weder Diskette noch CD-ROM mit den Beispiels-Programmen. Eine ideale Zweitlektüre für den VBA-Programmierer.

Ein ausführlicher Ratgeber für das **Erstellen von Hilfe-Dateien** ist

Reinhard Engel, **Elektronisch publizieren mit dem Microsoft Help Workshop,** Microsoft Press, 1996

Engels Buch enthält auf einer beigelegten CD das für Windows 95 und Windows NT neu geschaffene Hilfe-Erstellungs-System „Help Workshop“, zahlreiche Beispiels-Programme sowie eine ausgezeichnete Druckformat-Vorlage für das Schreiben von Hilfe-Dateien unter WinWord.

Hinweis EXCEL 7.0 teilt dem Windows-Hilfesystem dummerweise mit, daß beim Klicken auf den Schaltknop „Inhalt“ einer selbstgeschaffenen Hilfe-Datei das Inhaltsverzeichnis der EXCEL-Hilfe anzuzeigen ist, und nicht die Titelseite der selbstgeschaffenen Hilfe-Datei. In CompuServe ist aber die Datei SHOWHELP.XLS veröffentlicht worden; sie enthält ein Makro-Programm, das beim Hilfe-Aufruf diesen Fehler ausgleicht.

Hilfe Online

Per Telefon Microsoft gibt unter der aus ganz Österreich zum Ortstarif anwählbaren Telefonnummer 0660 6511 nach dem ersten Anruf 90 Tage lang gratis Auskunft. Beim ersten Anruf werden Sie registriert—es macht nur Sinn, wenn man EXCEL legal benützt!

CompuServe Unter dem Namen „MSCE“ erreichen Sie Microsoft Central Europe. Dort erhalten Sie ohne jede zeitliche Einschränkung, gratis und ohne nach Ihrer Lizenz gefragt zu werden Auskunft. Sie können auch Ihren nicht funktionierenden Code überspielen und erhalten meistens auch kurze Beispiele für funktionierenden Code zurück. „MSEXCE“ ist eine amerikanische CompuServe-Adresse für VBA-Fragen und -Lösungen.

© by Josef Broukal, Vienna, Austria

Internet Unter der Adresse „http://www.microsoft.com“ gibt es Gratis-Programme und die „Knowledge Base“, in der alle von Microsoft offiziell zur Kenntis genommenen Fehler und Ungereimtheiten in VBA für EXCEL enthalten sind. Manchmal wird auch ein Weg gezeigt, der den Fehler umgeht.

Hilfe durch J. Broukal

Ich helfe Ihnen auch gerne weiter, kann dafür allerdings nicht viel Zeit aufbringen. Sie erreichen mich unter (Wien) 87878 2217. Am liebsten ist mir, Sie schicken mir das Programm, das ich mir ansehen soll, an meine Internet-Mail-Adresse „jb@via.at“ zu. Eine Diskette mit frankiertem Rück–Kuvert an den ORF, 1136 Wien, tut’s auch.

Dieses Skriptum soll durch Feedback von Lesern und durch regelmäßiges Durchkämmen von EXCEL-VBA-user groups im Internet wachsen. Es wird sich also lohnen, von Zeit zu Zeit an der Internet-Adresse „http://www.joanneum.ac.at/services/vbaexcel“ nachzusehen.

© by Josef Broukal, Vienna, Austria